aws

User Guide

# Amazon ElastiCache



**API Version 2015-02-02**

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

# Amazon ElastiCache: 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 ElastiCache? ..... 1**
Serverless caching ..... 1
Node-based clusters ..... 2
Related services ..... 2
How it works ..... 3
Cache and caching engines ..... 3
Choosing between deployment options ..... 9
ElastiCache resources ..... 16
AWS Regions and Availability Zones ..... 18
Use Cases ..... 20
In-Memory Data Store ..... 21
Gaming Leaderboards ..... 22
Messaging (Pub/Sub) ..... 23
Recommendation Data (Hashes) ..... 26
Semantic caching for generative AI applications ..... 27
Durability for system-of-record workloads ..... 29
ElastiCache Customer Testimonials ..... 29
Semantic caching ..... 30
Agentic memory ..... 48
**Getting started with ElastiCache ..... 58**
Setting up ElastiCache ..... 58
Sign up for an AWS account ..... 59
Set up permissions ..... 59
Set up EC2 ..... 60
Grant network access ..... 60
Set up command line access ..... 61
Create a Valkey serverless cache ..... 62
With a public endpoint ..... 63
In a VPC ..... 76
Create a Redis OSS serverless cache ..... 80
Read and write data ..... 82
Clean up ..... 83
Next Steps ..... 84
Create a Memcached serverless cache ..... 84

Read and write data ... 86
Clean up ... 91
Next Steps ... 92
Network type for serverless caches ... 92
Configuring subnets for network type ... 93
Using the AWS Management Console ... 93
Using the AWS CLI ... 93
Tutorials: Getting started with Python and ElastiCache ... 94
Python and ElastiCache ... 95
Tutorial: Configuring Lambda to access ElastiCache in a VPC ... 114
Step 1: Creating an ElastiCache serverless cache. ... 114
Step 2: Create a Lambda function for ElastiCache ... 117
Step 3: Test the Lambda function with ElastiCache ... 121
Step 4: Clean up (Optional) ... 121
**Agent tools ... 123**
ElastiCache agent skills ... 124
Getting started with ElastiCache skill ... 124
How the ElastiCache skill works ... 126
Guardrails and tracking usage ... 129
Valkey MCP Server ... 131
Getting started with Valkey MCP ... 131
How Valkey MCP Server works ... 133
Guardrails and tracking usage ... 135
**Migrating to ElastiCache ... 136**
Online migration to ElastiCache Serverless ... 136
How a migration works ... 137
Migration states ... 138
Monitoring a migration ... 141
How you learn that a migration finished ... 144
Supported sources and targets ... 145
Errors returned by the migration operations ... 145
Prerequisites ... 146
Source authentication ... 160
Migrate your data ... 167
Cutover modes ... 175
Online migration to ElastiCache node-based clusters ... 182

Overview ... 182
Migration steps ... 183
Preparing your source and target for migration ... 183
Testing the data migration ... 185
Starting migration ... 185
Verifying the data migration progress ... 187
Completing the data migration ... 188
Performing online data migration using the Console ... 188
**Creating and managing a node-based ElastiCache cluster ... 191**
Components and features ... 191
Nodes ... 192
ElastiCache shards ... 192
ElastiCache clusters ... 193
ElastiCache replication ... 195
ElastiCache endpoints ... 197
Parameter groups ... 198
ElastiCache security ... 198
Subnet groups ... 199
ElastiCache backups ... 199
Events ... 199
ElastiCache terminology ... 200
Tutorial: How to create a node-based ElastiCache cluster ... 202
Creating a node-based ElastiCache cluster for Valkey ... 202
Creating a node-based ElastiCache cluster for Redis OSS ... 222
Deleting a cluster ... 242
Other tutorials and videos ... 244
Videos ... 245
Managing nodes in ElastiCache ... 250
Viewing ElastiCache Node Status ... 251
Valkey or Redis OSS nodes and shards ... 255
Connecting to nodes ... 258
Supported node types ... 263
Rebooting nodes ... 291
Replacing nodes (Valkey and Redis OSS) ... 296
Replacing nodes (Memcached) ... 302
Reserved nodes ... 303

Migrating previous generation nodes ... 319
Managing clusters in ElastiCache ... 322
Choosing a network type in ElastiCache ... 325
Auto Discovery (Memcached) ... 330
Data tiering in ElastiCache ... 374
Preparing a cluster in ElastiCache ... 380
Creating a cluster for Valkey or Redis OSS ... 391
Creating a cluster for Memcached ... 400
Viewing an ElastiCache cluster's details ... 403
Modifying an ElastiCache cluster ... 416
Adding nodes to an ElastiCache cluster ... 422
Removing nodes from an ElastiCache cluster ... 433
Canceling pending add or delete node operations in ElastiCache ... 442
Deleting a cluster in ElastiCache ... 443
Accessing your ElastiCache cluster or replication group ... 446
Finding connection endpoints in ElastiCache ... 455
Shards in ElastiCache ... 470
Comparing node-based Valkey, Memcached, and Redis OSS clusters ... 476
Choosing regions and availability zones for ElastiCache ... 480
Availability Zone considerations with Memcached ... 481
Locating your nodes ... 484
Supported Regions & endpoints ... 484
Using Local zones with ElastiCache ... 489
Using Outposts with ElastiCache ... 491
**Working with ElastiCache ... 495**
Snapshot and restore ... 495
Constraints ... 496
Performance impact of backups of node-based clusters ... 497
Scheduling automatic backups ... 498
Taking manual backups ... 500
Creating a final backup ... 507
Describing backups ... 510
Copying backups ... 512
Exporting a backup ... 514
Restoring from a backup ... 522
Deleting a backup ... 524

Tagging backups ... 526
Tutorial: Seeding a node-based cluster with a backup ... 527
Engine versions and upgrading in ElastiCache ... 537
Upgrading engine versions ... 537
Extended Support ... 542
Version Management for ElastiCache ... 546
Valkey 9.1 ... 548
Valkey 9.0 ... 548
Valkey 8.2 ... 550
Valkey 8.1 ... 550
Valkey 8.0 ... 550
ElastiCache version 7.2.6 for Valkey ... 551
Supported Redis OSS versions ... 551
Redis OSS versions end of life schedule ... 565
Supported Memcached versions ... 567
Major version behavior and compatibility differences for Valkey ... 571
Major engine version behavior and compatibility differences with Redis OSS ... 572
Upgrade considerations when working with node-based clusters ... 576
Best practices and caching strategies ... 578
Overall best practices ... 579
Best Practices for using Read Replicas ... 580
Supported and restricted Valkey, Memcached, and Redis OSS commands ... 584
Valkey and Redis OSS configuration and limits ... 625
IPv6 client examples for Valkey, Memcached, and Redis OSS ... 628
Best practices for clients (Valkey and Redis OSS) ... 629
Best practices for clients (Memcached) ... 654
TLS enabled dual stack ElastiCache clusters ... 658
Managing reserved memory for Valkey and Redis OSS ... 661
Best practices when working with Valkey and Redis OSS node-based clusters ... 663
Caching database query results ... 669
Caching strategies for Memcached ... 671
E-commerce caching ... 677
Durability ... 686
Durability options ... 687
Configuring durability ... 689
Changing durability on an existing cluster ... 690

Consistency ... 693
Failure scenarios ... 693
Limitations ... 694
Monitoring ... 694
Getting started with durability ... 695
Managing your node-based cluster in ElastiCache ... 696
Auto Scaling Valkey and Redis OSS clusters ... 697
Modifying cluster mode ... 743
Replication across AWS Regions using global datastores ... 746
High availability using replication groups ... 774
Managing ElastiCache cluster maintenance ... 860
Configuring engine parameters using ElastiCache parameter groups ... 863
Connecting an EC2 instance and an ElastiCache cache automatically ... 979
Automatic EC2 connectivity ... 980
Viewing connected compute resources ... 986
Scaling ElastiCache ... 987
Considerations ... 987
Scaling ElastiCache Serverless clusters ... 988
Scaling node-based clusters ... 991
Getting started with Bloom filters ... 1074
Bloom data type ... 1075
Bloom size limit ... 1076
ACLs ... 1076
Bloom-related metrics ... 1077
Bloom filter commands ... 1077
Getting started with Watch in Serverless ... 1078
Watch limit ... 1080
Supported commands related to Watch ... 1080
Getting started with Search ... 1080
Search features and limits ... 1081
Choosing the appropriate configuration ... 1085
Search write throttling ... 1086
Getting started with data aggregations ... 1089
Getting started with JSON for Valkey and Redis OSS ... 1090
JSON data type overview ... 1090
JSON commands ... 1102

Tagging your ElastiCache resources ... 1145
Monitoring costs with tags ... 1156
Managing tags using the AWS CLI ... 1158
Managing tags using the ElastiCache API ... 1163
Amazon ElastiCache Well-Architected Lens ... 1166
Operational Excellence Pillar ... 1167
Security Pillar ... 1175
Reliability Pillar ... 1181
Performance Efficiency Pillar ... 1187
Cost Optimization Pillar ... 1197
Troubleshooting in ElastiCache ... 1203
Connection issues ... 1204
Valkey or Redis OSS client errors ... 1205
Troubleshooting high latency in ElastiCache Serverless ... 1205
Troubleshooting throttling issues in ElastiCache Serverless ... 1207
Persistent connection issues ... 1208
Related Topics ... 1229
**Security ... 1230**
Data protection ... 1231
Data security in Amazon ElastiCache ... 1232
Internetwork traffic privacy ... 1325
Amazon VPCs and ElastiCache security ... 1326
ElastiCache API and interface VPC endpoints (AWS PrivateLink) ... 1351
Subnets and subnet groups ... 1356
Identity and Access Management ... 1366
Audience ... 1366
Authenticating with identities ... 1366
Managing access using policies ... 1368
How Amazon ElastiCache works with IAM ... 1369
Identity-based policy examples ... 1375
Troubleshooting ... 1378
Access control ... 1379
Overview of managing access ... 1381
Compliance validation ... 1431
More information ... 1432
Resilience ... 1432

Mitigating Failures ... 1433
Infrastructure security ... 1438
Service updates ... 1438
Managing service updates ... 1438
Addressed security vulnerabilities ... 1444
CVEs addressed in Amazon ElastiCache ... 1444
CVEs that do not affect Amazon ElastiCache ... 1450
**Logging and monitoring ... 1451**
Metrics and events for Valkey and Redis OSS serverless ... 1451
Serverless metrics ... 1452
Serverless events ... 1463
Metrics and events for node-based Valkey and Redis OSS clusters ... 1476
OpenTelemetry metrics ... 1476
Classic CloudWatch metrics ... 1476
Node-based cluster events (Valkey and Redis OSS) ... 1477
Monitoring ElastiCache with OpenTelemetry metrics ... 1485
How OpenTelemetry metrics work ... 1485
Metric attributes ... 1486
Supported configurations ... 1486
Turning on detailed monitoring ... 1486
Viewing metrics in the ElastiCache Insights dashboard ... 1489
Querying your metrics ... 1489
Pricing ... 1497
OpenTelemetry metrics reference for Amazon ElastiCache ... 1497
Example queries for common monitoring tasks ... 1528
Metrics and events for Memcached ... 1544
Memcached serverless metrics ... 1544
Memcached serverless events ... 1548
Node-based Memcached metrics ... 1557
Node-based Memcached events ... 1561
Logging Amazon ElastiCache API calls with AWS CloudTrail ... 1561
Amazon ElastiCache information in CloudTrail ... 1561
Understanding Amazon ElastiCache log file entries ... 1562
Amazon SNS event monitoring ... 1566
Managing ElastiCache Amazon SNS notifications ... 1566
Viewing ElastiCache events ... 1576

Event Notifications and Amazon SNS ... 1580
Log delivery ... 1587
Contents of a slow log entry ... 1588
Contents of an engine log entry ... 1589
Permissions to configure logging ... 1589
Log type and log format specifications ... 1590
ElastiCache logging destinations ... 1591
Specifying log delivery using the Console ... 1594
Specifying log delivery using the AWS CLI ... 1595
Monitoring use ... 1600
Host-Level Metrics ... 1601
Metrics for Valkey and Redis OSS ... 1605
Metrics for Memcached ... 1625
Which Metrics Should I Monitor? ... 1631
Choosing Metric Statistics and Periods ... 1636
Monitoring CloudWatch Cluster and Node Metrics ... 1636
**Quotas ... 1640**
**Reference ... 1643**
Using the ElastiCache API ... 1643
Using the query API ... 1643
Available libraries ... 1647
Troubleshooting applications ... 1647
Set up the AWS CLI for ElastiCache ... 1648
Prerequisites ... 1649
Getting the command line tools ... 1650
Setting up the tools ... 1651
Providing credentials for the tools ... 1652
Environmental variables ... 1653
Error messages ... 1654
Notifications ... 1656
General ElastiCache notifications ... 1656
ElastiCache for Memcached notifications ... 1656
ElastiCache for Redis OSS specific notifications ... 1657
**ElastiCache Documentation history ... 1658**
**AWS Glossary ... 1694**

# What is Amazon ElastiCache?

Welcome to the *Amazon ElastiCache User Guide*. Amazon ElastiCache is a web service that makes it easy to set up, manage, and scale a distributed in-memory data store or cache environment in the cloud. It provides a high-performance, scalable, and cost-effective caching solution. At the same time, it helps remove the complexity associated with deploying and managing a distributed cache environment.

You can operate Amazon ElastiCache in two formats. You can get started with a serverless cache or create a node-based cluster.

> **Note**
>
> Amazon ElastiCache works with the Valkey, Memcached, and Redis OSS engines. If you're unsure which engine you want to use, see Comparing node-based Valkey, Memcached, and Redis OSS clusters in this guide.
> To get started with ElastiCache using AI agents, visit the Agent tools for ElastiCache page to install the ElastiCache Skill and Valkey MCP server, which give AI agents the specialized knowledge and tools needed to work with ElastiCache accurately.

## Serverless caching

ElastiCache offers serverless caching, which simplifies adding and operating a cache for your application. ElastiCache Serverless enables you to create a highly available cache in under a minute, and eliminates the need to provision instances or configure nodes or clusters. Developers can create a Serverless cache by specifying the cache name using the ElastiCache console, SDK or CLI.

ElastiCache Serverless also removes the need to plan and manage caching capacity. ElastiCache constantly monitors the cache's memory, compute, and network bandwidth used by your application, and scales to meet the needs of your application. ElastiCache offers a simple endpoint experience for developers, by abstracting the underlying cache infrastructure and cluster design. ElastiCache manages hardware provisioning, monitoring, node replacements, and software patching automatically and transparently, so that you can focus on application development, rather than operating the cache.

ElastiCache Serverless supports two connection types. You can create a cache with the VPC or Public connection type. With a public endpoint, you can connect to your cache over the internet without a VPC. Public endpoints require IAM authentication and TLS 1.3 for every connection. You choose the connection type when you create the cache. For more information, see the section called “With a public endpoint”.

ElastiCache Serverless is compatible with Valkey 7.2 and later, Memcached 1.6.22 and later, and Redis OSS 7.1. Public endpoints are available on ElastiCache Serverless caches that use Valkey 9.0 or later.

## Creating a node-based cluster

If you need fine-grained control over your ElastiCache cluster, you can choose to create a node-based Valkey, Memcached, or Redis OSS cluster. ElastiCache enables you to create a node-based cluster by choosing the node-type, number of nodes, and node placement across AWS Availability Zones for your cluster. Since ElastiCache is a fully-managed service, it automatically manages hardware provisioning, monitoring, node replacements, and software patching for your cluster.

Creating a node-based cluster offers greater flexibility and control over your clusters. For example, you can choose to operate a cluster with single-AZ availability or multi-AZ availability depending on your needs. You can also choose to run Valkey, Memcached, or Redis OSS in cluster mode enabling horizontal scaling, or without cluster mode for just scaling vertically. When creating a node-based cluster, you are responsible for choosing the type and number of nodes correctly to ensure that your cache has enough capacity as required by your application. You can also choose when to apply new software patches to your Valkey or Redis OSS cluster.

When creating a node-based cluster you can choose from multiple supported versions of Valkey, Memcached and Redis OSS. For more information about supported engine versions see Engine versions and upgrading in ElastiCache.

For node-based Valkey clusters, you can enable *durability* to persist your data in a distributed Multi-AZ transactional log. With durability enabled, your data is protected even if all cache nodes fail, and replicas recover independently without impacting primary node performance. For more information, see the section called “Durability”.

## Related services

MemoryDB

When deciding whether to use ElastiCache or MemoryDB consider the following comparisons:

- ElastiCache is a service that is commonly used to cache data from other databases and data stores using Valkey, Memcached, or Redis OSS. You should consider ElastiCache for caching workloads where you want to accelerate data access with your existing primary database or data store (microsecond read and write performance). You should also consider ElastiCache for use cases where you want to use Valkey or Redis OSS data structures and APIs to access data stored in a primary database or data store.
- ElastiCache can also help you save database costs by storing frequently accessed data in a cache. If your application has high read throughput requirements, you can achieve high scale, fast performance, and lowered data storage costs by using ElastiCache, instead of scaling your underlying database.
- With *durability enabled*, ElastiCache for Valkey can also serve as a durable datastore with microsecond read latency and single-digit millisecond write latency (synchronous writes) or microsecond write latency (asynchronous writes). For more information, see the section called "Durability".
- MemoryDB is a durable, in-memory database for workloads that require an ultra-fast, primary database. It is compatible with Valkey and Redis OSS. You should consider using MemoryDB if your workload requires *multi-Region active-active replication with conflict-free data types (CRDTs)*. For single-Region durable workloads, consider using ElastiCache with durability enabled.

Amazon Relational Database Service

For further background information on the related Amazon Relational Database Service service, see Amazon RDS

# How ElastiCache works

Here you can find an overview of the major components of an ElastiCache deployment.

## Cache and caching engines

A cache is an in-memory data store that you can use to store cached data. Typically, your application will cache frequently accessed data in a cache to optimize response times. ElastiCache offers two deployment options: serverless caches and node-based clusters. See Choosing between deployment options.

> **Note**
>
> Amazon ElastiCache works with the Valkey, Memcached, and Redis OSS engines. If you're unsure which engine you want to use, see Comparing node-based Valkey, Memcached, and Redis OSS clusters in this guide.

**Topics**


- How ElastiCache works
- Pricing dimensions
- ElastiCache backups


## How ElastiCache works

### ElastiCache Serverless

ElastiCache Serverless enables you to create a cache without worrying about capacity planning, hardware management, or cluster design. You simply provide a name for your cache and you receive a single endpoint that you can configure in your Valkey, Memcached, Redis OSS client to begin accessing your cache.

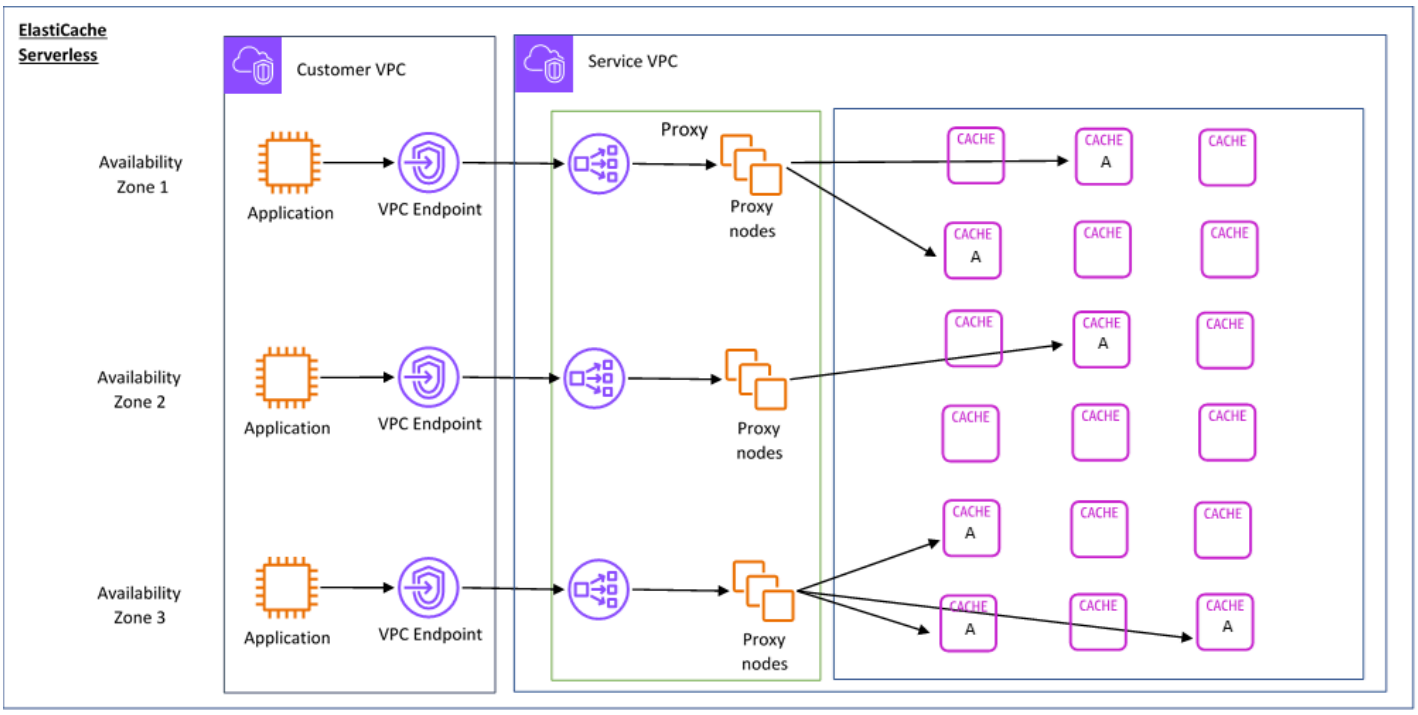
> **Note**
>
> - ElastiCache Serverless runs Valkey, Memcached, or Redis OSS in cluster mode and is only compatible with clients that support TLS.

**Key Benefits**

- **No capacity planning:** ElastiCache Serverless removes the need for you to plan for capacity. ElastiCache Serverless continuously monitors the memory, compute, and network bandwidth utilization of your cache and scales both vertically and horizontally. It allows a cache node to grow in size, while in parallel initiating a scale-out operation to ensure that the cache can scale to meet your application requirements at all times.
- **Pay-per-use:** With ElastiCache Serverless, you pay for the data stored and compute utilized by your workload on the cache. See Pricing dimensions.

- **High-availability:** ElastiCache Serverless automatically replicates your data across multiple Availability Zones (AZ) for high-availability. It automatically monitors the underlying cache nodes and replaces them in case of failures. It offers a 99.99% availability SLA for every cache.
- **Automatic software upgrades:** ElastiCache Serverless automatically upgrades your cache to the latest minor and patch software version without any availability impact to your application. When a new major version is available, ElastiCache will send you a notification.
- **Security:** Serverless always encrypts data in transit and at rest. You can use a service managed key or use your own Customer Managed Key to encrypt data at rest.

The following diagram illustrates how ElastiCache Serverless works.




When you create a new serverless cache, you choose a connection type. With a VPC endpoint, ElastiCache creates a Amazon Virtual Private Cloud (Amazon VPC) endpoint in the subnets of your choice, and your application connects to the cache through your VPC. With a public endpoint, your application connects to the cache directly over the internet, without a VPC.

With ElastiCache Serverless you receive a single DNS endpoint that your application connects to. When you request a new connection to the endpoint, ElastiCache Serverless handles all cache connections through a proxy layer. The proxy layer helps reduce complex client configuration, because the client does not need to rediscover cluster topology in case of changes to the

underlying cluster. The proxy layer is a set of proxy nodes that handle connections using a network load balancer.

When your application creates a new cache connection, the request is sent to a proxy node by the network load balancer. When your application executes cache commands, the proxy node that is connected to your application executes the requests on a cache node in your cache. The proxy layer abstracts the cluster topology and nodes from your client. This enables ElastiCache to intelligently load balance, scale out and add new cache nodes, replace cache nodes when they fail, and update software on the cache nodes, all without availability impact to your application or having to reset connections.

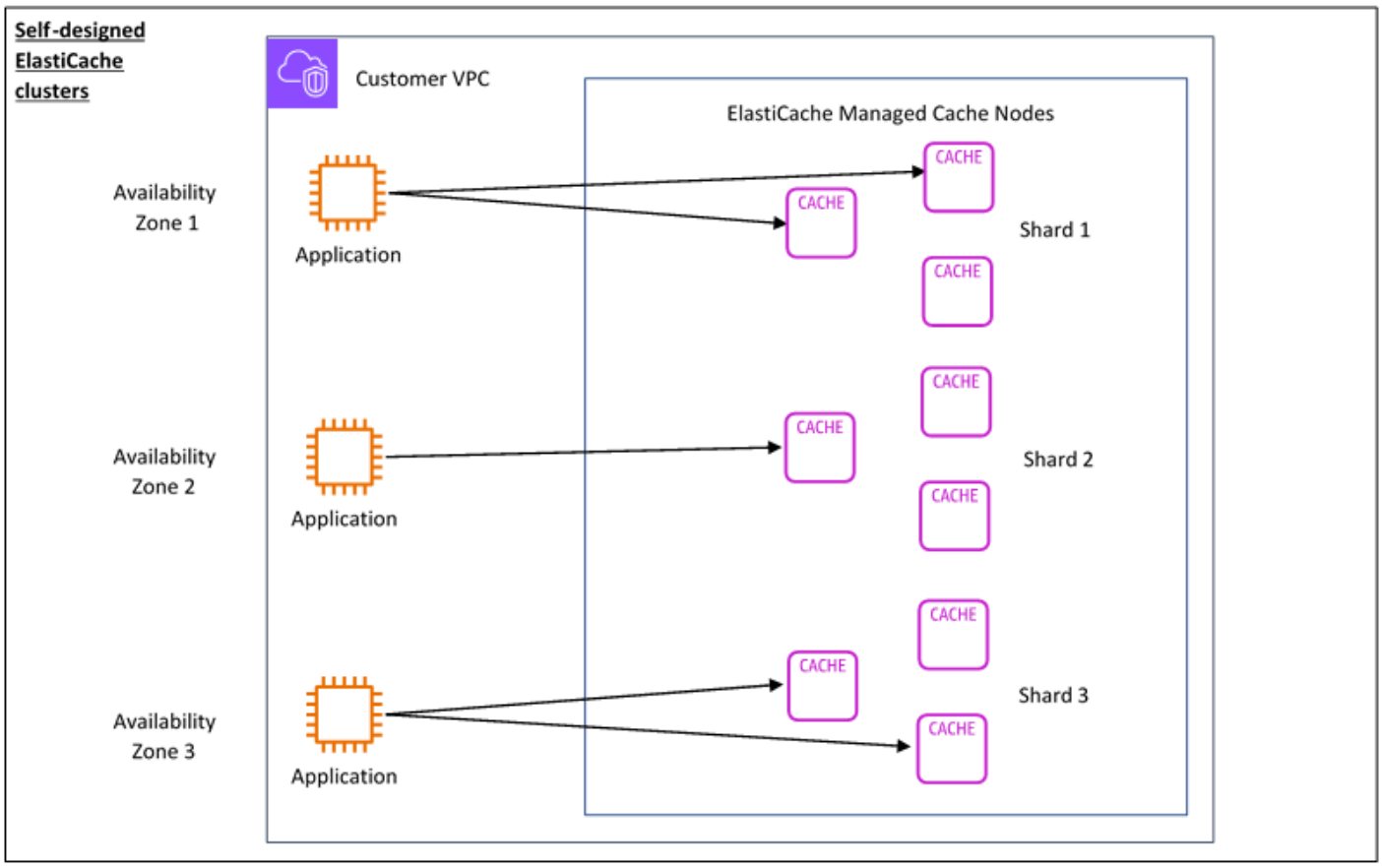
**Node-based clusters**

You can create a node-based ElastiCache cluster by choosing a cache node family, size, and number of nodes for your cluster. Creating a node-based cluster gives you finer grained control and enables you to choose the number of shards in your cache and the number of nodes (primary and replica) in each shard. You can choose to operate Valkey or Redis OSS in cluster mode by creating a cluster with multiple shards, or in non-cluster mode with a single shard.

**Key Benefits**

- **Create a node-based cluster:** With ElastiCache, you can create a node-based cluster and choose where you want to place your cache nodes. For example, if you have an application that wants to trade-off high-availability for low latency, you can choose to deploy your cache nodes in a single AZ. Alternatively, you can create a node-based cluster with nodes across multiple AZs to achieve high-availability.
- **Fine-grained control:** When creating a node-based cluster, you have more control over fine-tuning the settings on your cache. For example, you can use Valkey and Redis OSS parameters or Memcached specific parameters to configure the cache engine.
- **Scale vertically and horizontally:** You can choose to manually scale your cluster by increasing or decreasing the cache node size when needed. You can also scale horizontally by adding new shards or adding more replicas to your shards. You can also use the Auto-Scaling feature to configure scaling based on a schedule or scaling based on metrics like CPU and Memory usage on the cache.

The following diagram illustrates how node-based ElastiCache clusters work.




### Durability for node-based clusters

For node-based Valkey clusters (version 9.0 and higher), you can enable durability to persist your data in a Multi-AZ transactional log. With durability enabled, the architecture changes as follows:

- **Write path:** The primary node writes data to a Multi-AZ transactional log that replicates across at least two Availability Zones. With synchronous writes, data is acknowledged only after being persisted in the log. With asynchronous writes, data is acknowledged immediately and persisted asynchronously.
- **Read path (steady state):** In steady state, replicas continuously receive committed writes from the Multi-AZ transactional log and apply them in order. Reads on replicas are served immediately from in-memory state.
- **Replica recovery:** Replicas recover independently from the Multi-AZ transactional log and snapshots, without requiring any work from the primary node.

- **Off-box snapshotting:** Snapshots are created by ephemeral instances that read from the Multi-AZ transactional log, eliminating performance impact on customer workload.
- **Durability:** Data is recoverable even if all cache nodes fail, from the Multi-AZ transactional log and snapshots stored in Amazon S3.

## Pricing dimensions

You can deploy ElastiCache in two deployment options. When deploying ElastiCache Serverless, you pay for usage for data stored in GB-hours and compute in ElastiCache Processing Units (ECPU). When creating a node-based cluster, you pay per hour of the cache node usage. See pricing details here.

### Data storage

You pay for data stored in ElastiCache Serverless billed in gigabyte-hours (GB-hrs). ElastiCache Serverless continuously monitors the data stored in your cache, sampling multiple times per minute, and calculates an hourly average to determine the cache's data storage usage in GB-hrs. Each ElastiCache Serverless cache is metered for a minimum of 1 GB of data stored.

### ElastiCache Processing Units (ECPUs)

You pay for the requests your application executes on ElastiCache Serverless in ElastiCache Processing Units (ECPUs), a unit that includes both vCPU time and data transferred.

- Simple reads and writes require 1 ECPU for each kilobyte (KB) of data transferred. For example, a GET command that transfers up to 1 KB of data consumes 1 ECPU. A SET request that transfers 3.2 KB of data will consume 3.2 ECPUs.
- With Valkey and Redis OSS, commands that consume more vCPU time and transfer more data consume ECPUs based on the higher of the two dimensions. For example, if your application uses the HMGET command, consumes 3 times the vCPU time as a simple SET/GET command, and transfers 3.2 KB of data, it will consume 3.2 ECPU. Alternatively, if it transfers only 2 KB of data, it will consume 3 ECPUs.
- With Valkey and Redis OSS, commands that require additional vCPU time will consume proportionally more ECPUs. For example, if your application uses the Valkey or Redis OSS HMGET command, and consumes 3 times the vCPU time as a simple SET/GET command, then it will consume 3 ECPUs.
- With Memcached, commands that operate on multiple items will consume proportionally more ECPUs. For example, if your application performs a multiget on 3 items, it will consume 3 ECPUs.

- With Memcached, commands that operate on more items and transfer more data consume ECPUs based on the higher of the two dimensions. For example, if your application uses the GET command, retrieves 3 items, and transfers 3.2 KB of data, it will consume 3.2 ECPU. Alternatively, if it transfers only 2 KB of data, it will consume 3 ECPUs.

ElastiCache Serverless emits a new metric called `ElastiCacheProcessingUnits` that helps you understand the ECPUs consumed by your workload.

#### Node hours

You can create a node-based cluster by choosing the EC2 node family, size, number of nodes, and placement across Availability Zones. When creating a node-based cluster, you pay per hour for each cache node.

### ElastiCache backups

A *backup* is a point-in-time copy of a serverless cache, or a Valkey or Redis OSS node-based cluster. ElastiCache enables you to take a backup of your data at any time or setup automatic backups. Backups can be used to restore an existing cache or to seed a new cache. Backups consist of all the data in a cache plus some metadata. For more information see Snapshot and restore.

## Choosing between deployment options

Amazon ElastiCache has two deployment options:

- Serverless caching
- Node-based clusters

For a list of supported commands for both, see Supported and restricted Valkey, Memcached, and Redis OSS commands.

#### Serverless caching

Amazon ElastiCache Serverless simplifies cache creation and instantly scales to support customers' most demanding applications. With ElastiCache Serverless, you can create a highly-available and scalable cache in less than a minute, eliminating the need to provision, plan for, and manage cluster capacity. ElastiCache Serverless automatically stores data redundantly across three Availability Zones and provides a 99.99% availability Service Level Agreement (SLA). Backups from Valkey or Redis OSS node-based clusters can be restored into a serverless configuration.

### Node-based clusters

If you need fine-grained control over your Valkey, Memcached, or Redis OSS cluster, you can create a node-based cluster with ElastiCache. You choose the node-type, number of nodes, and node placement across AWS Availability Zones for your cluster. Since ElastiCache is a fully-managed service, it helps manage hardware provisioning, monitoring, node replacements, and software patching for your cluster. Node-based clusters can be designed to provide an up to 99.99% availability SLA. Backups from serverless Valkey or Redis OSS caches can be restored into a node-based cluster.

### Choosing between deployment options

Choose serverless caching if:

- You are creating a cache for workloads that are either new or difficult to predict.
- You have unpredictable application traffic.
- You want the easiest way to get started with a cache.
- You want to connect to your cache over the internet without a VPC, for example from a laptop, a serverless function, or a CI/CD pipeline.

Create your own node-based cluster if:

- You are already running ElastiCache Serverless and want finer grained control over the type of node running Valkey, Memcached, or Redis OSS, the number of nodes, and the placement of those nodes.
- You expect your application traffic to be relatively predictable, and you want fine-grained control over performance, availability, and cost.
- You can forecast your capacity requirements to control costs.
- You need durability for your data. When creating a node-based Valkey 9.0+ cluster, you can choose to enable durability through a Multi-AZ transactional log, enabling your cache to serve as a durable data store with zero data loss during failures.

## Comparing serverless caching and node-based clusters

| Feature | Serverless caching | Node-based clusters |
|---|---|---|
| Cache setup | Create a cache with just a name in under a minute | Provides fine-grained control over cluster design. User can choose node-type, number of nodes, and placement across AWS availability zones |
| Supported ElastiCache version | Valkey 7.2 and higher, Redis OSS version 7.1, Memcached 1.6.22 and higher | Valkey 7.2 and higher, Redis OSS version 4.0 to 7.1, Memcached 1.4 and higher |
| Cluster Mode (Valkey and Redis OSS) | Operates engines in `cluster mode enabled` only. Clients must support `cluster mode enabled` to connect to ElastiCache Serverless. | Can be configured to operate in cluster mode enabled or cluster mode disabled. |
| Scaling | Automatically scales engines both vertically and horizontally without any capacity management. | Provides control over scaling, while also requiring monitoring to make sure current capacity is adequately meeting demand.<br><br>For Valkey and Redis OSS, you can choose to scale vertically by increasing or decreasing the cache node size when needed. You can also scale horizontally, by adding new shards or adding more replicas to your shards. This capability is not available for Memcached. |

| Feature | Serverless caching | Node-based clusters |
| --- | --- | --- |
| | | With the Auto-Scaling feature you can also configure scaling based on a schedule, or scale based on metrics like CPU and Memory usage on the cache. |
| Client connection | Clients connect to a single endpoint. This enables the underlying cache node topology (scaling, replacements, and upgrades) to change without disconnecting the client. | Clients connect to each individual cache node. If a node is replaced, the client rediscovers cluster topology and re-establishes connections. |
| Public endpoints | Supported. Connect to your cache over the internet without a VPC. Available on Valkey 9.0 or later. | Not supported |
| Search | Supported. | Supported on Valkey 8.2 or later. |
| Configurability | No fine-grained configuration available. Customers can configure basic settings including subnets which can access the cache, whether automatic backups are turned on or off, and maximum cache usage limits. | Node-based clusters provide fine-grained configuration options. Customers can use parameter groups for fine-grained control. For a table of these parameter values by node type, see Engine specific parameters. |

| Feature | Serverless caching | Node-based clusters |
| --- | --- | --- |
| Multi-AZ | Data is replicated asynchron ously across multiple Availabil ity Zones for higher availabil ity and improved read latency. | Provides an option to create the cluster in a single Availability Zone or across multiple Availability Zones (AZs). When using Valkey or Redis OSS, provides Multi-AZ clusters with data replicate d asynchronously across multiple Availability Zones for higher availability and improved read latency. |
| Encryption at rest | Always enabled. Customers can use an AWS managed key or a customer managed key in AWS KMS. | Option to enable or disable encryption at rest. When enabled, customers can use an AWS managed key or a customer managed key in AWS KMS. |
| Encryption in transit (TLS) | Always enabled. Clients must support TLS connectivity. | Option to enable or disable. |

| Feature | Serverless caching | Node-based clusters |
| --- | --- | --- |
| Backups | Supports automatic and manual backups of caches with no performance impact.<br><br>Valkey and Redis OSS backups are cross-compatible, and can be restored into an ElastiCache Serverless cache or a node-based cluster. | Supports automatic and manual backups for Valkey and Redis OSS. Clusters may see some performance impact depending on the available reserved memory. For more information, see Managing reserved memory for Valkey and Redis OSS.<br><br>Valkey and Redis OSS backups are cross-compatible, and can be restored into an ElastiCache Serverless cache or a node-based cluster. |
| Monitoring | Support cache level metrics including cache hit rate, cache miss rate, data size, and ECPUs consumed.<br><br>ElastiCache Serverless sends events using EventBridge when significant events happen on your cache. You can choose to monitor, ingest, transform, and act on ElastiCache events using Amazon EventBridge. For more information, see Serverless cache events. | Node-based ElastiCache clusters emit metrics at each node level, including both host-level metrics and cache metrics.<br><br>Node-based clusters emit SNS notifications for significa nt events. See Metrics for Memcached and Metrics for Valkey and Redis OSS. |

| Feature | Serverless caching | Node-based clusters |
| --- | --- | --- |
| Availability | 99.99% availability Service Level Agreement (SLA) | Node-based clusters can be designed to achieve up to 99.99% availability Service Level Agreement (SLA), depending on the configuration. |
| Software upgrades and patching | Automatically upgrades cache software to the latest minor and patch version, without application impact. Customers receive a notification for major version upgrades, and customers can upgrade to the latest major version when they want. | Node-based clusters offer customer-enabled self-service for minor and patching version upgrades, as well as major version upgrades. Managed updates are automatically applied during customer defined maintenance windows. Customers can also choose to apply a minor or patch version upgrade on-demand. |
| Global Data Store | Not supported | Supports Global Data Store, which enables cross region replication with single region writes and multi-region reads |

| Feature | Serverless caching | Node-based clusters |
| --- | --- | --- |
| Data Tiering | Not supported | When you create a cluster with nodes from the r6gd, r7gd, or r8gd families, your data is tiered between memory and local solid state drive (SSD) storage. Data tiering provides a price-per formance option for Valkey and Redis OSS workloads by utilizing lower-cost solid state drives (SSDs) in each cluster node, in addition to storing data in memory. |
| Durability | Not supported | Supported for Valkey 9.0 and higher. Provides synchrono us or asynchronous writes through a Multi-AZ transacti onal log. |
| Pricing model | Pay-per-use, based on data stored in GB-hours and requests in ElastiCache Processing Units (ECPU). See pricing details here. | Pay-per-hour, based on cache node usage. See pricing details here. |

Related topics:

- Creating and managing a node-based ElastiCache cluster

## Amazon ElastiCache resources for first time users

We recommend that first time users begin by reading the following sections, and refer to them as needed.

- **Service highlights and pricing** – The product detail page provides a general product overview of ElastiCache, service highlights, and pricing.
- **ElastiCache videos** – The ElastiCache Videos section has videos that introduce you to Amazon ElastiCache. The videos cover common use cases for ElastiCache and demo how to use ElastiCache to reduce latency and improve throughput for your applications.
- **Getting started** – The Getting started with Amazon ElastiCache section includes information on creating a cache. It also includes how to authorize access to the cache, connect to a cache node, and delete the cache.
- **Performance at scale** – The Performance at scale with Amazon ElastiCache whitepaper addresses caching strategies that help your application to perform well at scale.

After you complete the preceding sections, read these sections:

- Choosing your node size

  You want your nodes to be large enough to accommodate all the data you want to cache. At the same time, you don't want to pay for more cache than you need. You can use this topic to help select the best node size.
- ElastiCache best practices and caching strategies

  Identify and address issues that can impact the efficiency of your cluster.

If you want to use the AWS Command Line Interface (AWS CLI), you can use these documents to help you get started:

- AWS Command Line Interface documentation

  This section provides information on downloading the AWS CLI, getting the AWS CLI working on your system, and providing your AWS credentials.
- AWS CLI documentation for ElastiCache

  This separate document covers all of the AWS CLI for ElastiCache commands, including syntax and examples.

You can write application programs to use the ElastiCache API with a variety of popular programming languages. Here are some resources:

- Tools for Amazon Web Services

  Amazon Web Services provides a number of software development kits (SDKs) with support for ElastiCache. You can code for ElastiCache using Java, .NET, PHP, Ruby, and other languages. These SDKs can greatly simplify your application development by formatting your requests to ElastiCache, parsing responses, and providing retry logic and error handling.

- Using the ElastiCache API

  If you don't want to use the AWS SDKs, you can interact with ElastiCache directly using the Query API. You can find troubleshooting tips and information on creating and authenticating requests and handling responses in this section.

- Amazon ElastiCache API Reference

  This separate document covers all of the ElastiCache API operations, including syntax and examples.

## AWS Regions and Availability Zones

Amazon cloud computing resources are housed in highly available data center facilities in different areas of the world (for example, North America, Europe, or Asia). Each data center location is called an AWS Region.

Each AWS Region contains multiple distinct locations called Availability Zones, or AZs. Each Availability Zone is engineered to be isolated from failures in other Availability Zones. Each is engineered to provide inexpensive, low-latency network connectivity to other Availability Zones in the same AWS Region. By launching instances in separate Availability Zones, you can protect your applications from the failure of a single location. For more information, see Choosing regions and availability zones.

You can create your cluster in several Availability Zones, an option called a Multi-AZ deployment. When you choose this option, Amazon automatically provisions and maintains a secondary standby node instance in a different Availability Zone. Your primary node instance is asynchronously replicated across Availability Zones to the secondary instance. This approach helps provide data redundancy and failover support, eliminate I/O freezes, and minimize latency spikes during system backups. For more information, see Minimizing downtime in ElastiCache for Valkey and Redis OSS with Multi-AZ.

With durability enabled, writes can be synchronously persisted to a Multi-AZ transactional log across at least two Availability Zones before being acknowledged, providing durability with zero data loss during failures.

# Common ElastiCache Use Cases and How ElastiCache Can Help

Whether serving the latest news, a top-10 leaderboard, a product catalog, or selling tickets to an event, speed is the name of the game. The success of your website and business is greatly affected by the speed at which you deliver content.

In "For Impatient Web Users, an Eye Blink Is Just Too Long to Wait," the New York Times noted that users can register a 250-millisecond (1/4 second) difference between competing sites. Users tend to opt out of the slower site in favor of the faster site. Tests done at Amazon, cited in How Webpage Load Time Is Related to Visitor Loss, revealed that for every 100-ms (1/10 second) increase in load time, sales decrease 1 percent.

If someone wants data, you can deliver that data much faster if it's cached. That's true whether it's for a webpage or a report that drives business decisions. Can your business afford to not cache your webpages so as to deliver them with the shortest latency possible?

It might seem intuitively obvious that you want to cache your most heavily requested items. But why not cache your less frequently requested items? Even the most optimized database query or remote API call is noticeably slower than retrieving a flat key from an in-memory cache. *Noticeably slower* tends to send customers elsewhere.

The following examples illustrate some of the ways using ElastiCache can improve overall performance of your application.

**Topics**


- In-Memory Data Store
- Gaming Leaderboards
- Messaging (Pub/Sub)
- Recommendation Data (Hashes)
- Semantic caching for generative AI applications
- Durability for system-of-record workloads
- ElastiCache Customer Testimonials
- Using Amazon ElastiCache for Valkey for semantic caching
- Using Amazon ElastiCache for Valkey for agentic memory

# In-Memory Data Store

The primary purpose of an in-memory key-value store is to provide ultrafast (submillisecond latency) and inexpensive access to copies of data. Most data stores have areas of data that are frequently accessed but seldom updated. Additionally, querying a database is always slower and more expensive than locating a key in a key-value pair cache. Some database queries are especially expensive to perform. An example is queries that involve joins across multiple tables or queries with intensive calculations. By caching such query results, you pay the price of the query only once. Then you can quickly retrieve the data multiple times without having to re-execute the query.

## What Should I Cache?

When deciding what data to cache, consider these factors:

**Speed and expense** – It's always slower and more expensive to get data from a database than from a cache. Some database queries are inherently slower and more expensive than others. For example, queries that perform joins on multiple tables are much slower and more expensive than simple, single table queries. If the interesting data requires a slow and expensive query to get, it's a candidate for caching. If getting the data requires a relatively quick and simple query, it might still be a candidate for caching, depending on other factors.

**Data and access pattern** – Determining what to cache also involves understanding the data itself and its access patterns. For example, it doesn't make sense to cache data that changes quickly or is seldom accessed. For caching to provide a real benefit, the data should be relatively static and frequently accessed. An example is a personal profile on a social media site. On the other hand, you don't want to cache data if caching it provides no speed or cost advantage. For example, it doesn't make sense to cache webpages that return search results because the queries and results are usually unique.

**Staleness** – By definition, cached data is stale data. Even if in certain circumstances it isn't stale, it should always be considered and treated as stale. To tell whether your data is a candidate for caching, determine your application's tolerance for stale data.

Your application might be able to tolerate stale data in one context, but not another. For example, suppose that your site serves a publicly traded stock price. Your customers might accept some staleness with a disclaimer that prices might be *n* minutes delayed. But if you serve that stock price to a broker making a sale or purchase, you want real-time data.

Consider caching your data if the following is true:

- Your data is slow or expensive to get when compared to cache retrieval.
- Users access your data often.
- Your data stays relatively the same, or if it changes quickly staleness is not a large issue.

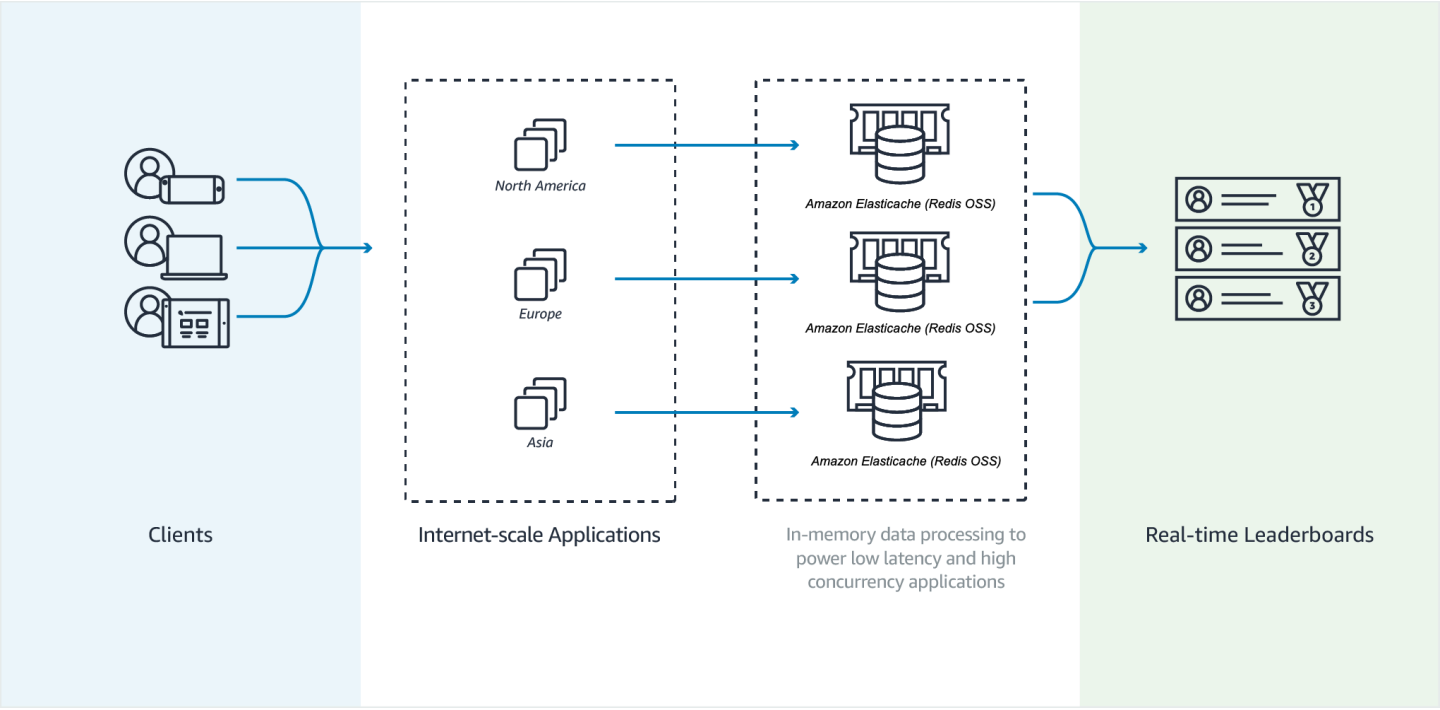
For more information, see Caching strategies for Memcached

## Gaming Leaderboards

With Valkey or Redis OSS sorted sets you can move the computational complexity of leaderboards from your application to your cluster.

Leaderboards, such as the top 10 scores for a game, are computationally complex. This is especially true when there is a large number of concurrent players and continually changing scores. Valkey and Redis OSS sorted sets guarantee both uniqueness and element ordering. With sorted sets, each time a new element is added to the sorted set it's reranked in real time. It's then added to the set in its correct numeric order.

In the following diagram, you can see how an ElastiCache gaming leaderboard works.




### Example Valkey or Redis OSS Leaderboard

In this example, four gamers and their scores are entered into a sorted list using ZADD. The command ZREVRANGEBYSCORE lists the players by their score, high to low. Next, ZADD is used to

update June's score by overwriting the existing entry. Finally, ZREVRANGEBYSCORE lists the players by their score, high to low. The list shows that June has moved up in the rankings.

```
ZADD leaderboard 132 Robert
ZADD leaderboard 231 Sandra
ZADD leaderboard 32 June
ZADD leaderboard 381 Adam

ZREVRANGEBYSCORE leaderboard +inf -inf
1) Adam
2) Sandra
3) Robert
4) June

ZADD leaderboard 232 June

ZREVRANGEBYSCORE leaderboard +inf -inf
1) Adam
2) June
3) Sandra
4) Robert
```

The following command tells June where she ranks among all the players. Because ranking is zero-based, *ZREVRANK* returns a 1 for June, who is in second position.

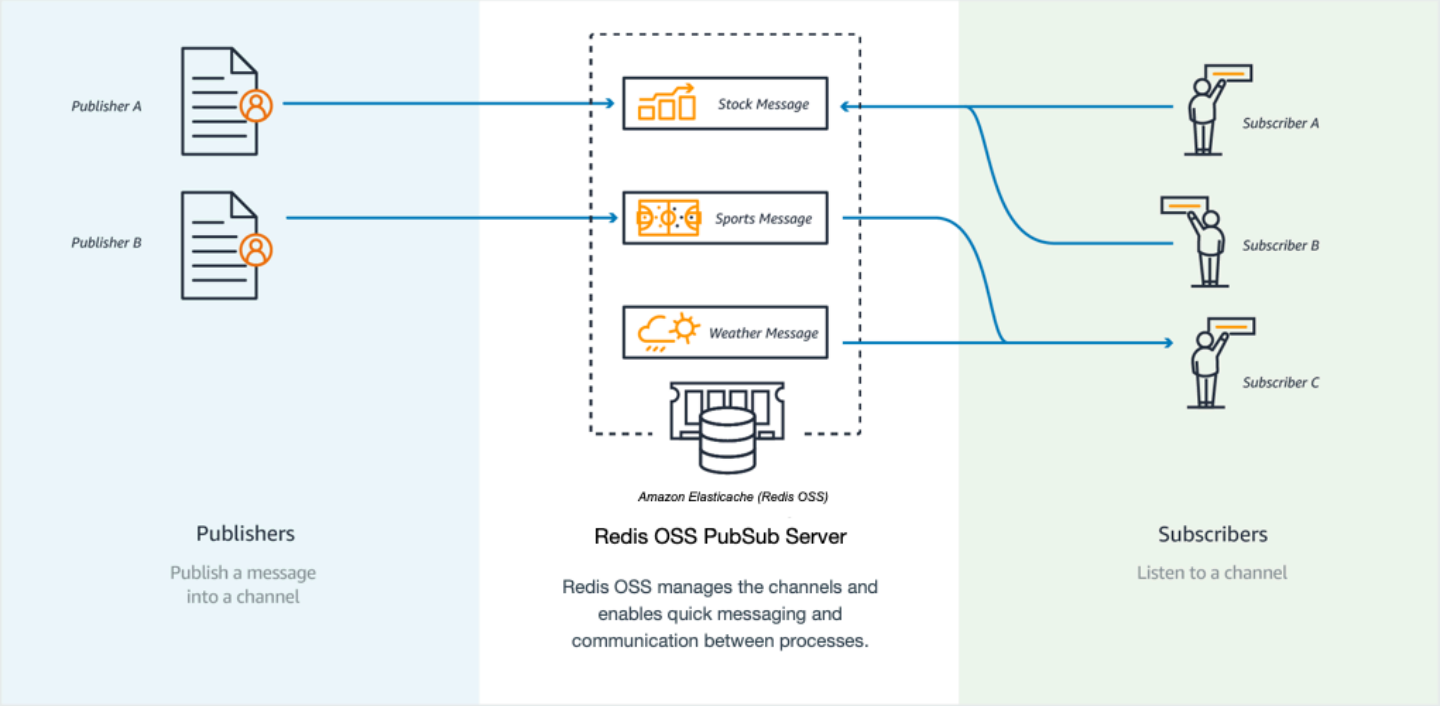
```
ZREVRANK leaderboard June
1
```

For more information, see the Valkey documentation about sorted sets.

## Messaging (Pub/Sub)

When you send an email message, you send it to one or more specified recipients. In the Valkey and Redis OSS pub/sub paradigm, you send a message to a specific channel not knowing who, if anyone, receives it. The people who get the message are those who are subscribed to the channel. For example, suppose that you subscribe to the *news.sports.golf* channel. You and all others subscribed to the *news.sports.golf* channel get any messages published to *news.sports.golf*.

Pub/sub functionality has no relation to any key space. Therefore, it doesn't interfere on any level. In the following diagram, you can find an illustration of ElastiCache messaging with Valkey and Redis OSS.




## Subscribing

To receive messages on a channel, you subscribe to the channel. You can subscribe to a single channel, multiple specified channels, or all channels that match a pattern. To cancel a subscription, you unsubscribe from the channel specified when you subscribed to it. Or, if you subscribed using pattern matching, you unsubscribe using the same pattern that you used before.

### Example- Subscription to a Single Channel

To subscribe to a single channel, use the SUBSCRIBE command specifying the channel you want to subscribe to. In the following example, a client subscribes to the *news.sports.golf* channel.

```
SUBSCRIBE news.sports.golf
```

After a while, the client cancels their subscription to the channel using the UNSUBSCRIBE command specifying the channel to unsubscribe from.

```
UNSUBSCRIBE news.sports.golf
```

### Example- Subscriptions to Multiple Specified Channels

To subscribe to multiple specific channels, list the channels with the SUBSCRIBE command. In the following example, a client subscribes to the *news.sports.golf*, *news.sports.soccer*, and *news.sports.skiing* channels.

```
SUBSCRIBE news.sports.golf news.sports.soccer news.sports.skiing
```

To cancel a subscription to a specific channel, use the UNSUBSCRIBE command and specify the channel to unsubscribe from.

```
UNSUBSCRIBE news.sports.golf
```

To cancel subscriptions to multiple channels, use the UNSUBSCRIBE command and specify the channels to unsubscribe from.

```
UNSUBSCRIBE news.sports.golf news.sports.soccer
```

To cancel all subscriptions, use UNSUBSCRIBE and specify each channel. Or use UNSUBSCRIBE and don't specify a channel.

```
UNSUBSCRIBE news.sports.golf news.sports.soccer news.sports.skiing
```

or

```
UNSUBSCRIBE
```

### Example- Subscriptions Using Pattern Matching

Clients can subscribe to all channels that match a pattern by using the PSUBSCRIBE command.

In the following example, a client subscribes to all sports channels. You don't list all the sports channels individually, as you do using SUBSCRIBE. Instead, with the PSUBSCRIBE command you use pattern matching.

```
PSUBSCRIBE news.sports.*
```

### Example Canceling Subscriptions

To cancel subscriptions to these channels, use the PUNSUBSCRIBE command.

```
PUNSUBSCRIBE news.sports.*
```

**Important**

- The channel string sent to a [P]SUBSCRIBE command and to the [P]UNSUBSCRIBE command must match. You can't PSUBSCRIBE to *news.** and PUNSUBSCRIBE from *news.sports.** or UNSUBSCRIBE from *news.sports.golf*.
- PSUBSCRIBE and PUNSUBSCRIBE are not available for ElastiCache Serverless.

### Publishing

To send a message to all subscribers to a channel, use the PUBLISH command, specifying the channel and the message. The following example publishes the message, "It's Saturday and sunny. I'm headed to the links." to the *news.sports.golf* channel.

```
PUBLISH news.sports.golf "It's Saturday and sunny. I'm headed to the links."
```

A client can't publish to a channel that it is subscribed to.

For more information, see Pub/Sub in the Valkey documentation.

## Recommendation Data (Hashes)

Using INCR or DECR in Valkey or Redis OSS makes compiling recommendations simple. Each time a user "likes" a product, you increment an *item:productID:like* counter. Each time a user "dislikes" a product, you increment an *item:productID:dislike* counter. Using hashes, you can also maintain a list of everyone who has liked or disliked a product.

#### Example- Likes and Dislikes

```
INCR item:38923:likes
HSET item:38923:ratings Susan 1
INCR item:38923:dislikes
HSET item:38923:ratings Tommy -1
```

# Semantic caching for generative AI applications

Operating generative AI applications at scale can be challenging due to the cost and latency associated with inference calls to large language models (LLMs). You can use ElastiCache for semantic caching in generative AI applications, allowing you to reduce the cost and latency of LLM inference calls. With semantic caching, you can return a cached response by using vector-based matching to find similarities between current and prior prompts. If a user's prompt is semantically similar to a prior prompt, a cached response will be returned instead of making a new LLM inference call, reducing the cost of generative AI applications and providing faster responses that improve the user experience. You can control which queries are routed to the cache by configuring similarity thresholds for prompts and applying tag or numeric metadata filters.

The inline real-time index updates provided by vector search for ElastiCache help ensure that the cache updates continuously as user prompts and LLM responses flow in. This real-time indexing is crucial to maintain freshness of cached results and cache hit rates, particularly for spiky traffic. In addition, ElastiCache simplifies operations for semantic caching through mature cache primitives such as per-key TTLs, configurable eviction strategies, atomic operations, and rich data structure and scripting support.

**Memory for generative AI assistants and agents**

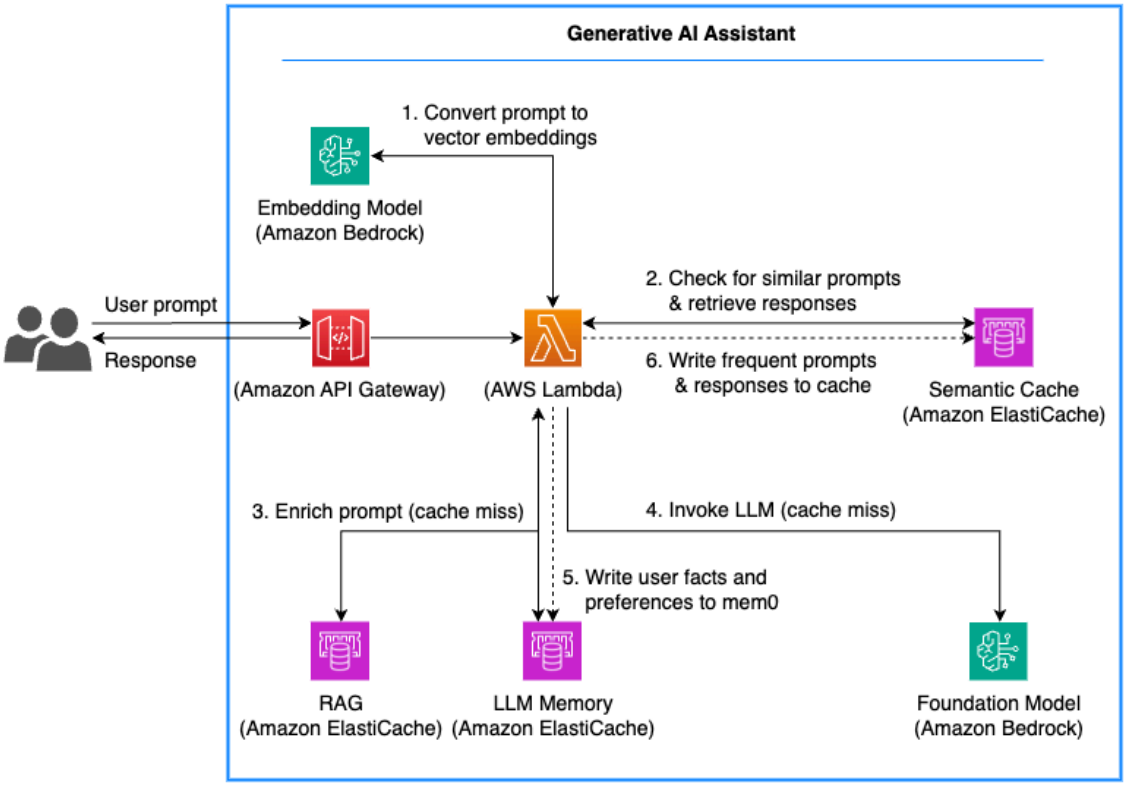
You can use ElastiCache to deliver more personalized, context-aware responses by implementing memory mechanisms that surface cross-session conversation history to LLMs. Conversational memory allows generative AI assistants and agents to retain and use past interactions to personalize responses and improve relevancy. However, simply aggregating all prior interactions into the prompt is ineffective since irrelevant extra tokens increase cost, degrade response quality and risk exceeding the LLM's context window. Instead, you can use vector search to retrieve and provide only the most relevant data in the context for each LLM invocation.

ElastiCache for Valkey provides integrations with open-source memory layers, providing built-in connectors to store and retrieve memories for LLM applications and agents. Vector search for ElastiCache provides fast index updates, keeping memory up to date and making new memories immediately searchable. Low latency vector search makes memory lookups fast, enabling them to be implemented in the online path of every request, not just background tasks. Beyond vector search, ElastiCache for Valkey also provides caching primitives for session state, user preferences, and feature flags, providing a single service to store short-lived session state and long-term "memories" in one datastore.

**Retrieval augmented generation (RAG)**

RAG is the process of providing LLMs with up-to-date information in the prompt to improve the relevance of responses. RAG reduces hallucinations and improves factual accuracy by grounding outputs with real-world data sources. RAG applications use vector search to retrieve semantically relevant content from a knowledge base. Low latency vector search provided by ElastiCache makes it suitable for implementing RAG in workloads that have large datasets with millions of vectors and above. Further, support for online vector index updates makes ElastiCache suitable for assistants with upload workflows that need to ensure any uploaded data is immediately searchable. RAG in agentic AI systems ensures that agents have up-to-date information for accurate actions. Low latency vector search is also crucial for RAG in agentic AI systems where a single query can trigger multiple LLM calls and stack up latency from underlying vector search.

The following diagram illustrates an example architecture using ElastiCache to implement a semantic cache, memory mechanisms, and RAG to enhance a generative AI application in production.




### Semantic Search

Vector search retrieves the most relevant text, speech, image, or video data based on closeness in meaning or features. This capability enables machine learning applications that rely on similarity search across diverse data modalities, including recommendation engines, anomaly detection, personalization, and knowledge management systems. Recommendation systems use vector

representations to capture complex patterns in user behavior and item characteristics, enabling them to suggest the most relevant content. Vector search for ElastiCache is well suited for these applications because of its near real-time updates and low latency, enabling similarity comparisons that deliver instant, highly relevant recommendations based on live user interactions.

## Durability for system-of-record workloads

If you enable durability and choose synchronous writes, you can use ElastiCache for use cases beyond caching where data loss is not acceptable. Examples include knowledge bases for RAG applications, AI agent memory, AI agent workflow state, payment tokenization, streaming metadata, and real-time inventory management.

## ElastiCache Customer Testimonials

To learn about how businesses like Airbnb, PBS, Esri, and others use Amazon ElastiCache to grow their businesses with improved customer experience, see How Others Use Amazon ElastiCache.

You can also watch the Tutorial videos for additional ElastiCache customer use cases.

# Using Amazon ElastiCache for Valkey for semantic caching

Large language models (LLMs) are the foundation for generative AI and agentic AI applications that power use cases from chatbots and search assistants to code generation tools and recommendation engines. As the use of AI applications in production grows, customers seek ways to optimize cost and performance. Most AI applications invoke the LLM for every user query, even when queries are repeated or semantically similar. Semantic caching is a method to reduce cost and latency in generative AI applications by reusing responses for identical or semantically similar requests using vector embeddings.

This topic explains how to implement a semantic cache using vector search on Amazon ElastiCache for Valkey, including the concepts, architecture, implementation, benchmarks, and best practices.

> **Note**
>
> To build semantic caching with AI agents, visit the Agent tools for ElastiCache page to install the ElastiCache Skill and Valkey MCP server. They provide the knowledge as well as automatic embedding generation and similarity search for setting up semantic caching workflows.

**Topics**


- Overview of semantic caching
- Why ElastiCache for Valkey for semantic caching
- Solution architecture
- Prerequisites
- Implementing a semantic cache with ElastiCache for Valkey
- Impact and benchmarks
- Multi-turn conversation caching
- Best practices
- Related resources

## Overview of semantic caching

Unlike traditional caches that rely on exact string matches, a semantic cache retrieves data based on semantic similarity. A semantic cache uses vector embeddings produced by models like Amazon Titan Text Embeddings to capture semantic meaning in a high-dimensional vector space.

In generative AI applications, a semantic cache stores vector representations of queries and their corresponding responses. The system compares the vector embedding of each new query against cached vectors of prior queries to determine if a similar query has been answered before. If the cache contains a similar query above a configured similarity threshold, the system returns the previously generated response instead of invoking the LLM. Otherwise, the system invokes the LLM to generate a response and caches the query embedding and response together for future reuse.

**Why semantic, not exact match?**

Consider an IT help chatbot where thousands of users ask the same question. The following queries are different strings but carry the same meaning:

- "How do I install the VPN app on my laptop?"
- "Can you guide me through setting up the company VPN?"
- "Steps to get VPN working on my computer"

An exact-match cache treats each query as unique and invokes the LLM three times. A semantic cache recognizes these queries as semantically equivalent and returns the cached response for all three, invoking the LLM only once.

**Key benefits**

Semantic caching provides the following benefits for generative AI and agentic AI applications:

- **Reduced costs** – Reusing answers for similar questions reduces the number of LLM calls and overall inference spend. In benchmarks, semantic caching reduced LLM inference cost by up to 86%.
- **Lower latency** – Serving answers from the cache provides faster responses than running LLM inference. Cache hits return responses in milliseconds rather than seconds, achieving up to 88% latency reduction.
- **Improved scalability** – Reducing LLM calls for similar or repeated queries enables you to serve more requests within the same model throughput limits without increasing capacity.

- **Improved consistency** – Using the same cached response for semantically similar requests helps deliver a consistent answer for the same underlying question.

### Where semantic caching is effective

Semantic caching is particularly effective for the following types of applications:

| Application type | Description | Example |
|---|---|---|
| **RAG-based assistants and copilots** | Many queries are duplicate requests from different users against a shared knowledge base | IT help chatbot, product FAQ bot, documentation assistant |
| **Agentic AI applications** | Agents break tasks into multiple small steps that may repeatedly look up similar information | Compliance agent reusing policy lookups, research agent reusing prior findings |
| **Multimodal applications** | Matching similar audio segments, images, or video queries | Automated phone systems reusing guidance for repeated requests like store hours |

## Why ElastiCache for Valkey for semantic caching

Semantic caching workloads continuously write, search, and evict cache entries to serve the stream of incoming user queries while keeping responses fresh. The cache store must meet the following requirements:

- **Real-time vector updates** – New queries and responses must be immediately available in the cache to maintain hit rates.
- **Low-latency lookups** – The cache sits in the online request path of every query, so lookups must not add perceptible delay to end-user response time.
- **Efficient ephemeral management** – Entries are frequently written, read, and evicted, requiring efficient management of a hot set.

ElastiCache for Valkey meets these requirements:

- **Lowest latency vector search** – At the time of writing, ElastiCache for Valkey delivers the lowest latency vector search with the highest throughput and best price-performance at 95%+ recall rate among popular vector databases on AWS. Latency is as low as microseconds with up to 99% recall.
- **Multithreaded architecture** – Vector search on ElastiCache uses a multithreaded architecture that supports real-time vector updates and high write throughput while maintaining low latency for search requests.
- **Built-in cache features** – TTL (time to live), eviction policies (`allkeys-lru`), and atomic operations help manage the ephemeral hot set of entries that semantic caching creates.
- **Vector index support** – ElastiCache supports both HNSW (Hierarchical Navigable Small World) and FLAT index algorithms with COSINE, Euclidean, and inner product distance metrics.
- **Zero-downtime scalability** – ElastiCache supports scaling without downtime, allowing you to adjust capacity as your cache grows.
- **Framework integration** – ElastiCache for Valkey integrates with Amazon Bedrock AgentCore through the LangGraph framework, enabling you to implement a Valkey-backed semantic cache for agents built on Amazon Bedrock.

## Solution architecture

The following architecture implements a read-through semantic cache for an agent on Amazon Bedrock AgentCore. A request follows one of two paths:

- **Cache hit** – If ElastiCache finds a prior query above the configured similarity threshold, AgentCore returns the cached answer immediately. This path invokes only the embedding model and does not require LLM inference. This path has millisecond-level end-to-end latency and does not incur LLM inference cost.
- **Cache miss** – If no similar prior query is found, AgentCore invokes the LLM to generate a new answer and returns it to the user. The application then caches the prompt's embedding and answer in ElastiCache so that future similar prompts can be served from the cache.

## Prerequisites

To implement semantic caching with ElastiCache for Valkey, you need:

1. An AWS account with access to Amazon Bedrock, including Amazon Bedrock AgentCore Runtime, Amazon Titan Text Embeddings v2 model, and an LLM such as Amazon Nova Premier enabled in the US East (N. Virginia) Region.
2. The AWS Command Line Interface (AWS CLI) configured with Python 3.11 or later.
3. An Amazon Elastic Compute Cloud instance inside your Amazon VPC with the following packages installed:

```
pip install numpy pandas valkey bedrock-agentcore \
            langchain-aws 'langgraph-checkpoint-aws[valkey]'
```

4. An ElastiCache for Valkey cluster running version 8.2 or later, which supports vector search. For instructions on creating a cluster, see Creating a cluster for Valkey or Redis OSS.

## Implementing a semantic cache with ElastiCache for Valkey

The following walkthrough shows how to implement a read-through semantic cache using ElastiCache for Valkey with Amazon Bedrock.

### Step 1: Create an ElastiCache for Valkey cluster

Create an ElastiCache for Valkey cluster with version 8.2 or later using the AWS CLI:

```
aws elasticache create-replication-group \
  --replication-group-id "valkey-semantic-cache" \
  --cache-node-type cache.r7g.large \
  --engine valkey \
  --engine-version 8.2 \
  --num-node-groups 1 \
  --replicas-per-node-group 1
```

### Step 2: Connect to the cluster and configure embeddings

From your application code running on your Amazon EC2 instance, connect to the ElastiCache cluster and set up the embedding model:

```
from valkey.cluster import ValkeyCluster
from langchain_aws import BedrockEmbeddings

# Connect to ElastiCache for Valkey
```

```
valkey_client = ValkeyCluster(
    host="mycluster.xxxxxx.clustercfg.use1.cache.amazonaws.com",  # Your cluster
 endpoint
    port=6379,
    decode_responses=False
)

# Set up Amazon Bedrock Titan embeddings
embeddings = BedrockEmbeddings(
    model_id="amazon.titan-embed-text-v2:0",
    region_name="us-east-1"
)
```

Replace the host value with your ElastiCache cluster's configuration endpoint. For instructions on finding your cluster endpoint, see Accessing your ElastiCache cluster.

**Step 3: Create the vector index for the semantic cache**

Configure a ValkeyStore that automatically embeds queries using an HNSW index with COSINE distance for vector search:

```
from langgraph_checkpoint_aws import ValkeyStore
from hashlib import md5

store = ValkeyStore(
    client=valkey_client,
    index={
        "collection_name": "semantic_cache",
        "embed": embeddings,
        "fields": ["query"],           # Fields to vectorize
        "index_type": "HNSW",          # Vector search algorithm
        "distance_metric": "COSINE",   # Similarity metric
        "dims": 1024                   # Titan V2 produces 1024-d vectors
    }
)
store.setup()

def cache_key_for_query(query: str):
    """Generate a deterministic cache key for a query."""
    return md5(query.encode("utf-8")).hexdigest()
```

> **Note**
>
> ElastiCache for Valkey uses an index to provide fast and accurate vector search. The `FT.CREATE` command creates the underlying index. For more information, see Vector search for ElastiCache.

### Step 4: Implement cache search and update functions

Create functions to search the cache for semantically similar queries and to store new query-response pairs:

```
def search_cache(user_message: str, k: int = 3, min_similarity: float = 0.8):
    """Look up a semantically similar cached response from ElastiCache."""
    hits = store.search(
        namespace="semantic-cache",
        query=user_message,
        limit=k
    )
    if not hits:
        return None

    # Sort by similarity score (highest first)
    hits = sorted(hits, key=lambda h: h["score"], reverse=True)
    top_hit = hits[0]
    score = top_hit["score"]

    if score < min_similarity:
        return None  # Below similarity threshold

    return top_hit["value"]["answer"]  # Return cached answer


def store_cache(user_message: str, result_message: str):
    """Store a new query-response pair in the semantic cache."""
    key = cache_key_for_query(user_message)
    store.put(
        namespace="semantic-cache",
        key=key,
        value={
            "query": user_message,
            "answer": result_message
```

```
        }
    )
```

### Step 5: Implement the read-through cache pattern

Integrate the cache into your application's request handling:

```
import time

def handle_query(user_message: str) -> dict:
    """Handle a user query with read-through semantic cache."""
    start = time.time()

    # Step 1: Search the semantic cache
    cached_response = search_cache(user_message, min_similarity=0.8)

    if cached_response:
        # Cache hit - return cached response
        elapsed = (time.time() - start) * 1000
        return {
            "response": cached_response,
            "source": "cache",
            "latency_ms": round(elapsed, 1),
        }

    # Step 2: Cache miss - invoke LLM
    llm_response = invoke_llm(user_message)  # Your LLM invocation function

    # Step 3: Store the response in cache for future reuse
    store_cache(user_message, llm_response)

    elapsed = (time.time() - start) * 1000
    return {
        "response": llm_response,
        "source": "llm",
        "latency_ms": round(elapsed, 1),
    }
```

### Underlying Valkey commands

The following table shows the Valkey commands used to implement the semantic cache:

| Operation | Valkey command | Typical latency |
|---|---|---|
| Create index | `FT.CREATE semantic_cache SCHEMA query TEXT answer TEXT embedding VECTOR HNSW 6 TYPE FLOAT32 DIM 1024 DISTANCE_METRIC COSINE` | One-time setup |
| Cache lookup | `FT.SEARCH semantic_cache "*=>[KNN 3 @embedding $query_vec]" PARAMS 2 query_vec [bytes] DIALECT 2` | Microseconds |
| Store response | `HSET cache:{hash} query "..." answer "..." embedding [bytes]` | Microseconds |
| Set TTL | `EXPIRE cache:{hash} 82800` | Microseconds |
| LLM inference (miss) | External API call to Amazon Bedrock | 500–6000 ms |

## Impact and benchmarks

AWS evaluated the approach on 63,796 real user chatbot queries and their paraphrased variants from the public SemBenchmarkLmArena dataset. This dataset captures user interactions with the Chatbot Arena platform across general assistant use cases such as question answering, writing, and analysis.

The evaluation used the following configuration:

- ElastiCache `cache.r7g.large` instance as the semantic cache store

- Amazon Titan Text Embeddings V2 for embeddings
- Claude 3 Haiku for LLM inference

The cache was started empty, and all 63,796 queries were streamed as random incoming user traffic, simulating real-world application traffic.

**Cost and accuracy at different similarity thresholds**

The following table summarizes the trade-off between cost reduction, latency improvement, and accuracy across different similarity thresholds:

| Similarity threshold | Cache hit ratio | Accuracy of cached responses | Total daily cost | Cost savings | Average latency (s) | Latency reduction |
|---|---|---|---|---|---|---|
| Baseline (no cache) | – | – | $49.50 | – | 4.35 | – |
| 0.99 (very strict) | 23.5% | 92.1% | $41.70 | 15.8% | 3.60 | 17.1% |
| 0.95 (strict) | 56.0% | 92.6% | $23.80 | 51.9% | 1.84 | 57.7% |
| 0.90 (moderate) | 74.5% | 92.3% | $13.60 | 72.5% | 1.21 | 72.2% |
| 0.80 (balanced) | 87.6% | 91.8% | $7.60 | 84.6% | 0.60 | 86.1% |
| 0.75 (relaxed) | 90.3% | 91.2% | $6.80 | **86.3%** | 0.51 | **88.3%** |
| 0.50 (very relaxed) | 94.3% | 87.5% | $5.90 | 88.0% | 0.46 | 89.3% |

At a similarity threshold of 0.75, semantic caching reduced LLM inference cost by up to 86% while maintaining 91% answer accuracy. The choice of LLM, embedding model, and backing store affects

both cost and latency. Semantic caching delivers proportionally larger benefits when used with bigger, higher-cost LLMs.

**Individual query latency improvements**

The following table shows the impact on individual query latency. A cache hit reduced latency by up to 59x, from multiple seconds to a few hundred milliseconds:

| Query intent | Cache miss latency | Cache hit latency | Reduction |
|---|---|---|---|
| "Are there instances where SI prefixes deviate from denoting powers of 10, excluding their application?" → paraphrased variant | 6.51 s | 0.11 s | **59x** |
| "Sally is a girl with 3 brothers, and each of her brothers has 2 sisters. How many sisters are there in Sally's family?" → paraphrased variant | 1.64 s | 0.13 s | **12x** |

## Multi-turn conversation caching

For applications with multi-turn conversations, the same user message can mean different things depending on context. For example, "Tell me more" in a conversation about Valkey means something different from "Tell me more" in a conversation about Python.

**The challenge**

Single-prompt caching works well for stateless queries. In multi-turn conversations, you must cache the full conversation context, not just the last message:

```
# "Tell me more" means nothing without context
```

```
# Conversation A: "What is Valkey?" -> "Tell me more"  (about Valkey)
# Conversation B: "What is Python?" -> "Tell me more"  (about Python)
```

**Strategy: context-aware cache keys**

Instead of embedding only the last user message, embed a summary of the full conversation context. This way, similar follow-up questions in similar conversation flows can reuse cached answers.

```
def build_context_string(messages: list) -> str:
    """Build a cacheable context string from conversation messages."""
    # Use last 3 turns (6 messages: user + assistant pairs)
    recent = messages[-6:]
    parts = []
    for msg in recent:
        role = msg["role"]
        content = msg["content"][:200]  # Truncate long messages
        parts.append(f"{role}: {content}")
    return " | ".join(parts)
```

**Per-user cache isolation with TAG filters**

Use TAG fields to isolate cached conversations by user, session, or other dimensions. This prevents one user's cached conversations from being returned for another user:

```
# Create index with TAG field for per-user isolation
valkey_client.execute_command(
    "FT.CREATE", "conv_cache_idx",
    "SCHEMA",
    "context_summary", "TEXT",
    "response", "TEXT",
    "user_id", "TAG",
    "turn_count", "NUMERIC",
    "embedding", "VECTOR", "HNSW", "6",
    "TYPE", "FLOAT32",
    "DIM", "1024",
    "DISTANCE_METRIC", "COSINE",
)
```

Search with hybrid filtering (TAG + KNN):

```
def lookup_conversation_cache(messages: list, user_id: str, threshold: float = 0.12):
```

```
    """Search cache for similar conversation contexts, scoped to a user.

    Note: FT.SEARCH with COSINE distance returns a distance score where
    0 = identical and 2 = opposite. A lower score means higher similarity.
    The threshold here is a maximum distance: only return results closer
    than this value.
    """
    context = build_context_string(messages)
    query_vec = get_embedding(context)

    # Hybrid search: filter by user_id TAG + KNN on context embedding
    results = valkey_client.execute_command(
        "FT.SEARCH", "conv_cache_idx",
        f"@user_id:{{{user_id}}}=>[KNN 1 @embedding $query_vec]",
        "PARAMS", "2", "query_vec", query_vec,
        "DIALECT", "2",
    )

    if results[0] > 0:
        fields = results[2]
        field_dict = {fields[j]: fields[j+1] for j in range(0, len(fields), 2)}
        distance = float(field_dict.get("__embedding_score", "999"))
        if distance < threshold:  # Lower distance = more similar
            return {"hit": True, "response": field_dict.get("response", ""),
"distance": distance}

    return {"hit": False}
```

**Note**

The `@user_id:{user_123}` TAG filter ensures that User A's cached conversations don't leak to User B. The hybrid query (TAG + KNN) runs as a single atomic operation — pre-filtering by user, then finding the nearest conversation context.

### Cache isolation strategies

| Strategy | TAG filter | Best for |
|---|---|---|
| Per-user | `@user_id:{user_123}` | Personalized assistants |

| Strategy | TAG filter | Best for |
|---|---|---|
| Per-session | `@session_id:{sess_abc}` | Short-lived chats |
| Global (shared) | No filter (*) | FAQ bots, common queries |
| Per-model | `@model:{gpt-4}` | Multi-model deployments |
| Per-product | `@product_id:{prod_456}` | E-commerce assistants |

## Best practices

### Choosing data that can be cached

Semantic caching is well suited for repeated queries whose responses are relatively stable, whereas real-time or highly dynamic responses are often poor candidates for caching.

Use tag and numeric filters derived from existing application context (such as product ID, category, region, or user segment) to decide which queries and responses are eligible for caching and to improve the relevance of cache hits.

### Similarity threshold tuning

The similarity threshold controls the trade-off between cache hit rate and answer quality. Choose a threshold that balances cost savings with accuracy for your use case:

| Threshold | Hit rate | Quality risk | Best for |
|---|---|---|---|
| 0.95 (strict) | Low (~25%) | Very low | Medical, legal, financial applications |
| 0.90 (moderate) | Medium (~55%) | Low | General chatbots |
| 0.80 (balanced) | High (~75%) | Low–Medium | FAQ bots, IT support |
| 0.75 (relaxed) | Very high (~90%) | Medium | High-volume repetitive queries |

> **Important**
>
> Start with a higher threshold (0.90–0.95) and gradually lower it while monitoring accuracy. Use A/B testing to find the optimal balance for your workload.

### Standalone queries versus conversations

- **For standalone queries** – Apply semantic caching directly on the user query text.
- **For multi-turn conversations** – First use your conversation memory to retrieve the key facts and recent messages needed to answer the current turn. Then apply semantic caching to the combination of the current user message and the retrieved context, instead of embedding the entire raw dialogue.

### Setting cache invalidation periods

Use TTL to control how long cached responses are served before they are regenerated on a cache miss.

| Data type | Recommended TTL | Rationale |
|---|---|---|
| Static facts (documentation, policies) | 24 hours | Facts change infrequently |
| Product information | 12–24 hours | Updated daily in most catalogs |
| General assistant responses | 1–4 hours | Balance freshness with hit rate |
| Real-time data (prices, inventory) | 5–15 minutes | Data changes frequently |
| Conversation context | 30 minutes | Session-scoped, short-lived |

```
# Set TTL with random jitter to spread out cache invalidations
import random
```

```
base_ttl = 82800  # ~23 hours
jitter = random.randint(0, 3600)  # Up to 1 hour of jitter
valkey_client.expire(cache_key, base_ttl + jitter)
```

**Tip**

Set TTLs that match your application use case and how often your data or model outputs change. Longer TTLs increase cache hit rates but raise the risk of outdated answers. Shorter TTLs keep responses fresher but lower cache hit rates and require more LLM inference.

### Monitoring and cost tracking

Track cache performance metrics to optimize your semantic cache over time:

```
def record_cache_event(valkey_client, event_type: str):
    """Track cache hits and misses using atomic counters."""
    valkey_client.incr(f"cache:metrics:{event_type}")

    # Also track hourly for time-series analysis
    from datetime import datetime
    hour_key = datetime.now().strftime("%Y%m%d%H")
    counter_key = f"cache:metrics:{event_type}:{hour_key}"
    valkey_client.incr(counter_key)
    valkey_client.expire(counter_key, 86400 * 7)  # Keep 7 days

def get_cache_stats(valkey_client) -> dict:
    """Get current cache performance metrics."""
    hits = int(valkey_client.get("cache:metrics:hit") or 0)
    misses = int(valkey_client.get("cache:metrics:miss") or 0)
    total = hits + misses
    hit_rate = hits / total if total > 0 else 0

    avg_cost_per_call = 0.015  # Example: ~$0.015 per LLM call
    savings = hits * avg_cost_per_call

    return {
        "total_requests": total,
        "hits": hits,
        "misses": misses,
        "hit_rate": round(hit_rate, 3),
```

```
        "estimated_savings_usd": round(savings, 2),
    }
```

### Memory management

- **Set maxmemory policy** – Configure `maxmemory-policy allkeys-lru` on your ElastiCache cluster to automatically evict least-recently-used cache entries when the cluster reaches its memory limit.
- **Plan for capacity** – Each cache entry typically requires approximately 4–6 KB (embedding dimensions × 4 bytes + query text + response text). A 1 GB ElastiCache instance can store approximately 170,000 cached entries.
- **Use cache invalidation for stale data** – When underlying data changes, use text search to find and invalidate related cache entries:

```
def invalidate_by_topic(valkey_client, topic_keyword: str):
    """Remove cached entries matching a topic after a data update."""
    results = valkey_client.execute_command(
        "FT.SEARCH", "semantic_cache",
        f"@query:{topic_keyword}",
        "NOCONTENT",  # Only return keys, not fields
    )

    if results[0] > 0:
        keys = results[1:]
        for key in keys:
            valkey_client.delete(key)
        print(f"Invalidated {len(keys)} cached entries for '{topic_keyword}'")
```

## Related resources

- Search for ElastiCache
- Common ElastiCache use cases
- Lower cost and latency for AI using Amazon ElastiCache as a semantic cache with Amazon Bedrock (AWS Database Blog)
- Announcing vector search for Amazon ElastiCache (AWS Database Blog)
- Amazon Bedrock AgentCore Runtime
- LangGraph checkpoint for Valkey

- Valkey client libraries

# Using Amazon ElastiCache for Valkey for agentic memory

Agentic AI applications use external tools, APIs, and multi-step reasoning to complete complex tasks. However, by default, agents don't retain memory between conversations, which limits their ability to provide personalized responses or maintain context across sessions. Amazon ElastiCache for Valkey provides the high-performance, low-latency infrastructure that agentic memory systems require to store, retrieve, and manage persistent memory for AI agents.

This topic explains how to use ElastiCache for Valkey as the storage layer for agentic memory, covering the concepts, architecture, implementation, and best practices for building memory-enabled AI agents.

> **Note**
>
> To build agent memory on ElastiCache with AI agents, visit the Agent tools for ElastiCache page to install the ElastiCache Skill and Valkey MCP server. They provide knowledge as well as tools to create agent memory workflows.

**Topics**


- Overview of agentic memory
- Types of agentic memory
- Why ElastiCache for Valkey for agentic memory
- Solution architecture
- Prerequisites
- Setting up ElastiCache for Valkey as a vector store for agentic memory
- Performance benefits
- Best practices
- Related resources


## Overview of agentic memory

An agentic AI application is a system that takes actions and makes decisions based on input. These agents use external tools, APIs, and multi-step reasoning to complete complex tasks. Without

persistent memory, agents forget everything between conversations, making it impossible to deliver personalized experiences or complete multi-step tasks effectively.

Agentic memory handles the persistence, encoding, storage, retrieval, and summarization of knowledge gained through user interactions. This memory system is a critical part of the context management component of an agentic AI application, enabling agents to learn from past conversations and apply that knowledge to future interactions.

Consider the following examples where agentic memory provides value:

- **Customer support agents** – An agent remembers a customer's previous issues, preferences, and account details across support sessions, avoiding repetitive information gathering and delivering faster resolutions.
- **Research agents** – An agent that researches GitHub repositories remembers previously discovered project metrics, avoiding redundant web searches and reducing token usage and response time.
- **Personal assistant agents** – An agent retains a user's scheduling preferences, communication style, and recurring tasks to provide increasingly personalized assistance over time.

## Types of agentic memory

### Short-term memory

Short-term memory maintains context within a single session. It tracks the current conversation flow, recent interactions, and intermediate reasoning steps. Short-term memory is essential for multi-turn conversations where the agent needs to reference earlier parts of the dialogue.

ElastiCache for Valkey supports short-term memory through data structures such as lists (for ordered chat history), hashes (for session metadata), and strings (for tool result caching with TTL-based expiration).

### Long-term memory

Long-term memory stores information across multiple sessions. This enables agents to remember user preferences, past decisions, and historical context for future conversations. Long-term memory requires a persistent, searchable store that supports semantic retrieval — finding relevant memories based on meaning rather than exact keyword matches.

ElastiCache for Valkey supports long-term memory through its vector similarity search capabilities (available in Valkey 8.2 and later). Vector search enables semantic memory retrieval, allowing

agents to find relevant memories based on meaning by comparing vector embeddings of stored memories against new queries.

**Additional memory types**

| Memory type | Description | ElastiCache support |
| --- | --- | --- |
| **Episodic memory** | Records of specific past interactions and events | Vector search over stored conversation embeddings |
| **Semantic memory** | General knowledge and facts extracted from interactions | Vector similarity search with HNSW or FLAT indexes |
| **Procedural memory** | Knowledge about how to perform tasks and use tools | Hash-based storage of tool configurations and workflows |

## Why ElastiCache for Valkey for agentic memory

ElastiCache for Valkey provides several capabilities that make it well suited as the storage layer for agentic memory:

- **Sub-millisecond latency** – ElastiCache for Valkey delivers microsecond-level latency for memory operations, making it suitable for real-time agent interactions where memory lookups must not add perceptible delay to the user experience.
- **Vector similarity search** – Starting with Valkey version 8.2, ElastiCache supports vector similarity search through the valkey-search module. This enables semantic memory retrieval, where agents can find relevant memories based on meaning rather than exact keyword matches.
- **Real-time index updates** – New memories become immediately searchable after being written. This is critical for agentic applications where the agent may need to recall information it stored moments ago within the same session.
- **Built-in cache management** – Features such as TTL (time to live), eviction policies (`allkeys-lru`), and atomic operations help manage the memory lifecycle.
- **Multiple data structures** – Valkey provides hashes, lists, strings, streams, JSON, and vectors — each optimized for different memory patterns. A single ElastiCache instance can support session state (hashes), conversation history (lists), tool result caching (strings with TTL), event logs (streams), and semantic memory (vectors).

- **Scalability** – ElastiCache scales to handle millions of requests with consistent low latency, supporting applications with large numbers of concurrent users and agents.

## Solution architecture

The following architecture implements persistent memory for agentic AI applications using ElastiCache for Valkey as the vector storage component.

**Key components:**

- **Amazon Bedrock AgentCore Runtime** – Provides the hosting environment for deploying and running agents. It provides access to the LLM and embedding models required for the architecture.
- **Agent framework (for example, Strands Agents)** – Manages LLM invocations, tool execution, and user conversations. Strands Agents supports multiple LLMs, including models from Amazon Bedrock, Anthropic, Google Gemini, and OpenAI.
- **Mem0** – The memory orchestration layer that sits between AI agents and storage systems. Mem0 manages the memory lifecycle, from extracting information from agent interactions to storing and retrieving it.
- **Amazon ElastiCache for Valkey** – The managed in-memory data store that serves as the vector storage component. ElastiCache uses Valkey's vector similarity search capabilities to store high-dimensional vector embeddings, enabling semantic memory retrieval.

## Prerequisites

To implement agentic memory with ElastiCache for Valkey, you need:

1. An AWS account with access to Amazon Bedrock, including Amazon Bedrock AgentCore Runtime and embedding models.
2. An ElastiCache cluster running Valkey 8.2 or later. Valkey 8.2 includes support for vector similarity search. For instructions on creating a cluster, see Creating a cluster for Valkey or Redis OSS.
3. An Amazon Elastic Compute Cloud instance or other compute resource within the same Amazon VPC as your ElastiCache cluster.
4. Python 3.11 or later with the following packages:

```
pip install strands-agents strands-agents-tools strands-agents-builder
pip install mem0ai "mem0ai[vector_stores]"
```

## Setting up ElastiCache for Valkey as a vector store for agentic memory

The following walkthrough shows how to build a memory-enabled AI agent using Mem0 with ElastiCache for Valkey as the vector store.

### Step 1: Create a basic agent without memory

First, install Strands Agents and create a basic agent:

```
pip install strands-agents strands-agents-tools strands-agents-builder
```

Initialize a basic agent with an HTTP tool for web browsing:

```
from strands import Agent
from strands.tools import http_request

# Initialize agent with access to the tool to browse the web
agent = Agent(tools=[http_request])

# Format messages as expected by Strands
formatted_messages = [
    {
        "role": "user",
        "content": [{"text": "What is the URL for the project mem0 and its most
 important metrics?"}]
    }
]

result = agent(formatted_messages)
```

Without memory, the agent performs the same research tasks repeatedly for each request. In testing, the agent makes three tool calls to answer the request, using approximately 70,000 tokens and taking over 9 seconds to complete.

### Step 2: Configure Mem0 with ElastiCache for Valkey

Install the Mem0 library with the Valkey vector store connector:

```
pip install mem0ai "mem0ai[vector_stores]"
```

Configure Valkey as the vector store. ElastiCache for Valkey supports vector search capabilities starting with version 8.2:

```
from mem0 import Memory

# Configure Mem0 with ElastiCache for Valkey
config = {
    "vector_store": {
        "provider": "valkey",
        "config": {
            "valkey_url": "your-elasticache-cluster.cache.amazonaws.com:6379",
            "index_name": "agent_memory",
            "embedding_model_dims": 1024,
            "index_type": "flat"
        }
    }
}

m = Memory.from_config(config)
```

Replace *your-elasticache-cluster.cache.amazonaws.com* with your ElastiCache cluster's endpoint. For instructions on finding your cluster endpoint, see Accessing your ElastiCache cluster.

**Step 3: Add memory tools to the agent**

Create memory tools that the agent can use to store and retrieve information. The `@tool` decorator transforms regular Python functions into tools the agent can invoke:

```
from strands import Agent, tool
from strands.tools import http_request

@tool
def store_memory_tool(information: str, user_id: str = "user") -> str:
    """Store important information in long-term memory."""
    memory_message = [{"role": "user", "content": information}]

    # Create new memories using Mem0 and store them in Valkey
    m.add(memory_message, user_id=user_id)
```

```
    return f"Stored: {information}"

@tool
def search_memory_tool(query: str, user_id: str = "user") -> str:
    """Search stored memories for relevant information."""

    # Search memories using Mem0 stored in Valkey
    results = m.search(query, user_id=user_id)
    if results['results']:
        return "\n".join([r['memory'] for r in results['results']])
    return "No memories found"

# Initialize Strands agent with memory tools
agent = Agent(tools=[http_request, store_memory_tool, search_memory_tool])
```

**Step 4: Test the memory-enabled agent**

With memory enabled, the agent stores information from its interactions and retrieves it in subsequent requests:

```
# First request - agent searches the web and stores results in memory
formatted_messages = [
    {
        "role": "user",
        "content": [{"text": "What is the URL for the project mem0 and its most
 important metrics?"}]
    }
]
result = agent(formatted_messages)

# Second request (same question) - agent retrieves from memory
result = agent(formatted_messages)
```

On the second request, the agent retrieves the information from memory instead of making web tool calls. In testing, this reduced token usage from approximately 70,000 to 6,300 (a 12x reduction) and improved response time from 9.25 seconds to 2 seconds (more than 3x faster).

**How it works under the hood**

The following table shows the Valkey commands that Mem0 uses internally to implement agentic memory with ElastiCache. Mem0 abstracts these commands through its API — the exact schema and key naming may vary depending on the Mem0 version and configuration:

| Operation | Valkey command | Description |
|---|---|---|
| Create vector index | `FT.CREATE agent_memory SCHEMA embedding VECTOR HNSW 6 TYPE FLOAT32 DIM 1024 DISTANCE_METRIC COSINE` | Creates a vector index for semantic memory search |
| Store memory | `HSET mem:{id} memory "..." embedding [bytes] user_id "user_123" created_at "..."` | Stores a memory with its vector embedding |
| Search memories | `FT.SEARCH agent_memory "*=>[KNN 5 @embedding $query_vec]" PARAMS 2 query_vec [bytes] DIALECT 2` | Finds the most semantically similar memories |
| Set expiration | `EXPIRE mem:{id} 86400` | Sets TTL for memory entries |

## Performance benefits

The following table summarizes the performance improvements observed in testing with a memory-enabled agent versus a stateless agent:

| Metric | Without memory | With memory | Improvement |
|---|---|---|---|
| Tool calls per request | 3 | 0 (memory retrieval) | Eliminated redundant tool calls |
| Token usage | ~70,000 | ~6,300 | **12x reduction** |
| Response time | 9.25 seconds | 2 seconds | **3x+ faster** |

| Metric | Without memory | With memory | Improvement |
| --- | --- | --- | --- |
| Memory lookup latency | N/A | Sub-millisecond | Valkey in-memory performance |

## Best practices

### Memory lifecycle management

- **Use TTL for short-term memory** – Set appropriate TTL values on memory entries to automatically expire transient information. For session context, use TTLs of 30 minutes to 24 hours. For long-term user preferences, use longer TTLs or persist indefinitely.
- **Implement memory decay** – Mem0 provides built-in decay mechanisms that remove irrelevant information over time. Configure these to prevent memory bloat as the agent accumulates more interactions.
- **Deduplicate memories** – Before storing a new memory, check if a similar memory already exists using vector similarity search. Update existing memories rather than creating duplicates.

### Vector index configuration

- **Choose the right index type** – Use `FLAT` for smaller memory stores (under 100,000 entries) where exact search is feasible. Use `HNSW` for larger stores where approximate nearest neighbor search provides better performance at scale.
- **Select appropriate dimensions** – Match the embedding dimensions to your model. Amazon Titan Text Embeddings V2 produces 1024-dimensional vectors. OpenAI's text-embedding-3-small produces 1536-dimensional vectors.
- **Use COSINE distance metric** – For text embeddings from models like Amazon Titan and OpenAI, COSINE distance is typically the most appropriate metric for measuring semantic similarity.

### Multi-user isolation

- **Scope memories by user ID** – Always include a `user_id` parameter when storing and searching memories to prevent information leaking between users.

- **Use TAG filters for efficient isolation** – When querying the vector index, use TAG filters (for example, `@user_id:{user_123}`) to pre-filter results by user before performing KNN search. This runs as a single atomic operation, providing both isolation and performance.

```
# Example: TAG-filtered vector search for user isolation
results = client.execute_command(
    "FT.SEARCH", "agent_memory",
    f"@user_id:{{{user_id}}}=>[KNN 5 @embedding $query_vec]",
    "PARAMS", "2", "query_vec", query_vec,
    "DIALECT", "2",
)
```

#### Memory management at scale

- **Set maxmemory policy** – Configure `maxmemory-policy allkeys-lru` on your ElastiCache cluster to automatically evict least-recently-used memory entries when the cluster reaches its memory limit.
- **Monitor memory usage** – Use Amazon CloudWatch metrics to track memory utilization, cache hit rates, and vector search latency. Set alarms for high memory usage to proactively manage capacity.
- **Plan for capacity** – Each memory entry typically requires approximately 4–6 KB (embedding dimensions × 4 bytes + metadata). A 1 GB ElastiCache instance can store approximately 170,000–250,000 memory entries depending on embedding size and metadata.

## Related resources

- Search for ElastiCache
- Common ElastiCache use cases
- Build persistent memory for agentic AI applications with Mem0 and Amazon ElastiCache for Valkey (AWS Database Blog)
- Mem0 documentation — Valkey vector store
- Strands Agents user guide
- Amazon Bedrock AgentCore Runtime
- Valkey vector search documentation

# Getting started with Amazon ElastiCache

Use the hands-on tutorial in this section to help you get started and learn more about using ElastiCache.

**Topics**


- Setting up ElastiCache
- Create a Valkey serverless cache
- Create a Redis OSS serverless cache
- Create a Memcached serverless cache
- Choosing a network type for serverless caches
- Tutorials: Getting started with Python and ElastiCache
- Tutorial: Configuring Lambda to access ElastiCache in a VPC


> **Note**
>
> To get started with ElastiCache using AI agents, visit the Agent tools for ElastiCache page to install the ElastiCache Skill and Valkey MCP server, which give AI agents the specialized knowledge and tools needed to work with ElastiCache accurately.

# Setting up ElastiCache

To use the ElastiCache web service, follow these steps.

> **Note**
>
> To create a cache with a public endpoint, see the section called “With a public endpoint”. A public endpoint lets you connect over the internet without setting up EC2, a VPC, or security groups.

**Topics**


- Sign up for an AWS account

- Set up your permissions (new ElastiCache users only)
- Set up EC2
- Grant network access from an Amazon VPC security group to your cache
- Download and set up command line access

## Sign up for an AWS account

To get started with AWS, you need an AWS account. There are two ways to sign up for AWS. If you want complete control over account configuration and where your content is stored, or if you have regulated workloads such as HIPAA or FedRAMP, use Sign up for AWS (advanced). If you are a new user and want to use a login you already own to sign in to AWS and create an AWS environment with preconfigured defaults in select AWS Regions, use Sign up for AWS.

## Set up your permissions (new ElastiCache users only)

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Amazon ElastiCache creates and uses service-linked roles to provision resources and access other AWS resources and services on your behalf. For ElastiCache to create a service-linked role for you, use the AWS-managed policy named `AmazonElastiCacheFullAccess`. This role comes preprovisioned with permission that the service requires to create a service-linked role on your behalf.

You might decide not to use the default policy and instead to use a custom-managed policy. In this case, make sure that you have either permissions to call `iam:createServiceLinkedRole` or that you have created the ElastiCache service-linked role.

For more information, see the following:

- Creating a New Policy (IAM)
- AWS managed policies for Amazon ElastiCache
- Using Service-Linked Roles for Amazon ElastiCache

## Set up EC2

You will need to setup an EC2 instance from which you will connect to your cache.

- If you don't already have an EC2 instance, learn how to setup an EC2 instance here: Amazon EC2 Getting Started Guide.
- Your EC2 instance must be in the same VPC and have the same security group settings as your cache. By default, Amazon ElastiCache creates a cache in your default VPC and uses the default security group. To follow this tutorial, ensure that your EC2 instance is in the default VPC and has the default security group.

## Grant network access from an Amazon VPC security group to your cache

ElastiCache node-based clusters use port 6379 for Valkey and Redis OSS commands, and ElastiCache serverless uses both port 6379 and port 6380. To successfully connect and execute Valkey or Redis OSS commands from your EC2 instance, your security group must allow access to these ports as needed.

For ElastiCache serverless specifically:

- **Port 6379 (Primary endpoint):** Required for write operations and reads requiring strong consistency
- **Port 6380 (Read-optimized endpoint):** Used for Read From Replica functionality, providing lower latency reads with eventual consistency

Many clients establish connections to both ports even if not actively using Read From Replica, requiring that you make port 6380 accessible even if you don't intend to use Read From Replica.

ElastiCache for Memcached uses the 11211 and 11212 ports to accept Memcached commands. To successfully connect and execute Memcached commands from your EC2 instance, your security group must allow access to these ports.

1. Sign in to the AWS Command Line Interface and open the Amazon EC2 console.
2. In the navigation pane, under **Network & Security**, choose **Security Groups**.
3. From the list of security groups, choose the security group for your Amazon VPC. Unless you created a security group for ElastiCache use, this security group will be named *default*.
4. Choose the Inbound tab, and then:
    a. Choose **Edit**.
    b. Choose **Add rule**.
    c. In the Type column, choose **Custom TCP rule**.
    d. If using Valkey or Redis OSS, then in the **Port range** box, type 6379.

       If using Memcached, then in the **Port range** box, type 11211.
    e. In the **Source** box, choose **Anywhere** which has the port range (0.0.0.0/0) so that any Amazon EC2 instance that you launch within your Amazon VPC can connect to your cache.
    f. If you are using ElastiCache serverless, add another rule by choosing **Add rule**.
    g. In the **Type** column, choose **Custom TCP** rule.
    h. If using ElastiCache for Redis OSS, then in the **Port range**box, type 6380.

       If using ElastiCache for Memcached, then in the **Port range**box, type 11212.
    i. In the **Source** box, choose **Anywhere** which has the port range (0.0.0.0/0) so that any Amazon EC2 instance that you launch within your Amazon VPC can connect to your cache.
    j. Choose **Save**

## Download and set up command line access

**Download and install the *valkey-cli* utility.**

If you use ElastiCache for Valkey, then you might find the valkey-cli utility useful. If you're using ElastiCache for Redis OSS with redis-cli, consider switching to valkey-cli as it works for Redis OSS as well.

1. Connect to your Amazon EC2 instance using the connection utility of your choice. For instructions on how to connect to an Amazon EC2 instance, see the Amazon EC2 Getting Started Guide.
2. Download and install the valkey-cli utility by running the following commands.

```
sudo dnf install gcc jemalloc-devel openssl-devel tcl tcl-devel -y
wget -O valkey-8.0.0.tar.gz https://github.com/valkey-io/valkey/archive/refs/
tags/8.0.0.tar.gz
tar xvzf valkey-8.0.0.tar.gz
cd valkey-8.0.0
make valkey-cli BUILD_TLS=yes
sudo install -m 755 src/valkey-cli /usr/local/bin/
```

**Note**

- When you install the redis6 package, it installs redis6-cli with default encryption support.
- It is important to have build support for TLS when installing valkey-cli or redis-cli. ElastiCache Serverless is only accessible when TLS is enabled.
- If you are connecting to a cluster that isn't encrypted, you don't need the `Build_TLS=yes` option.

# Create a Valkey serverless cache

When you create a Valkey serverless cache, you choose one of two connection types: VPC or Public. With the VPC connection type, which is the default, you access the cache privately from within your VPC. With the Public connection type, you access the cache over the internet through a public endpoint, without a VPC.

**Topics**

- Create a Valkey serverless cache with a public endpoint
- Create a Valkey serverless cache in a VPC

# Create a Valkey serverless cache with a public endpoint

With a public endpoint, you can connect to an ElastiCache Serverless cache directly over the internet without a VPC. Public endpoints are available on ElastiCache Serverless for Valkey 9.0 or later.

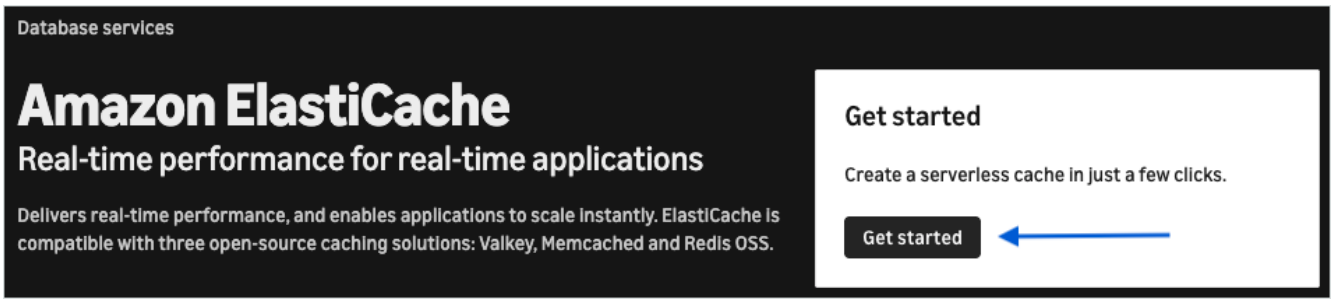
## Prerequisites

- An AWS account with permissions to create ElastiCache resources.
- Either `AdministratorAccess`, or IAM permissions that include `elasticache:CreateServerlessCache`. If you plan to create a custom user group instead of using the default, you also need `elasticache:CreateUser` and `elasticache:CreateUserGroup`.
- Your IAM identity must have permission to create caches with a public endpoint. This permission can be restricted using the `elasticache:ConnectionType` condition key. For more information, see the section called "Example Policies: Using Conditions for Fine-Grained Parameter Control".
- The AWS CLI version 2 installed and configured, for the CLI steps.

With ElastiCache, you can use the system-managed IAM user (`default.iam-user`) and user group (`default.iam-user-group`) to get started without creating custom users. To use a custom user group, first create a user group where all users use IAM authentication, then create the cache.

AWS Management Console

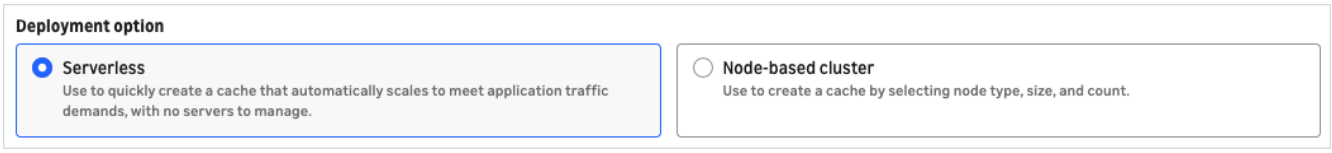
1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/. If prompted, sign in.
2. Choose **Get started** to open the Create cache page.



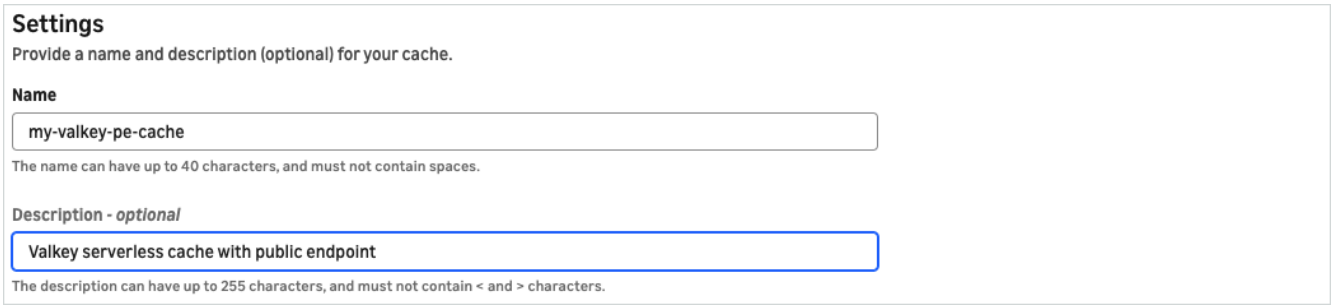

3. Under **Engine**, choose **Valkey**.




4. Under **Deployment option**, choose **Serverless**.



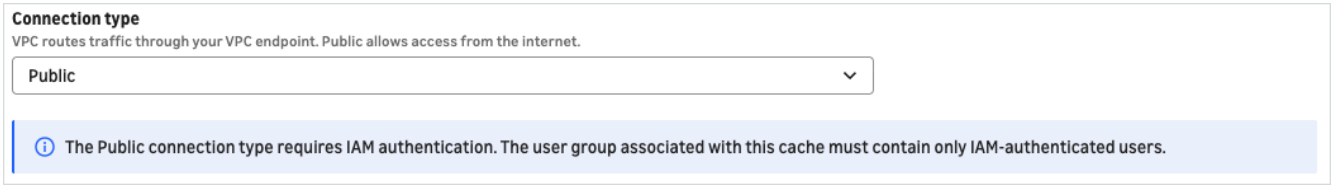

5. Under **Settings**, enter a **Name** for your cache. Optionally, enter a **Description**.



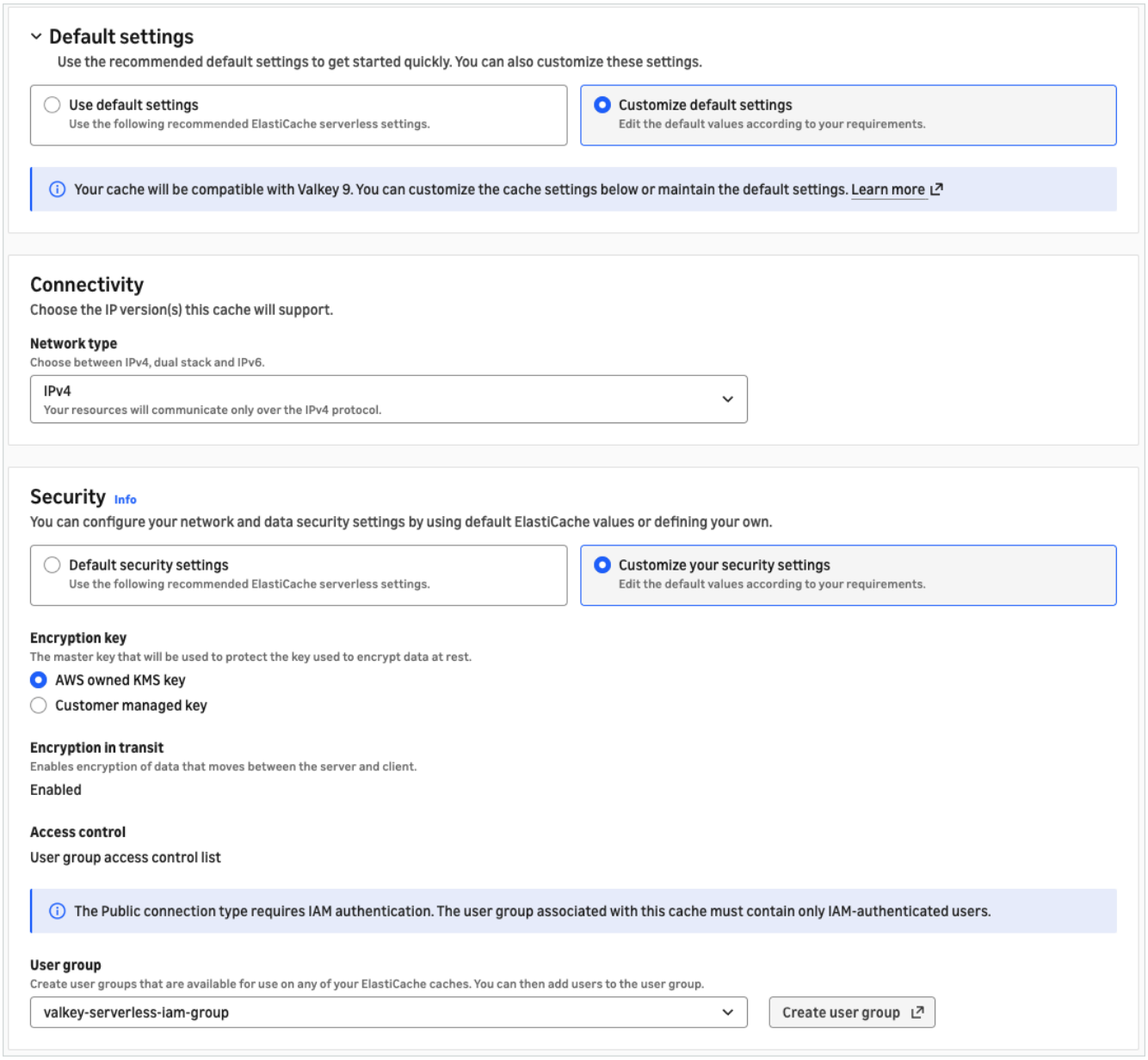

6. Under **Engine version**, keep the default selected or choose **9** or later.



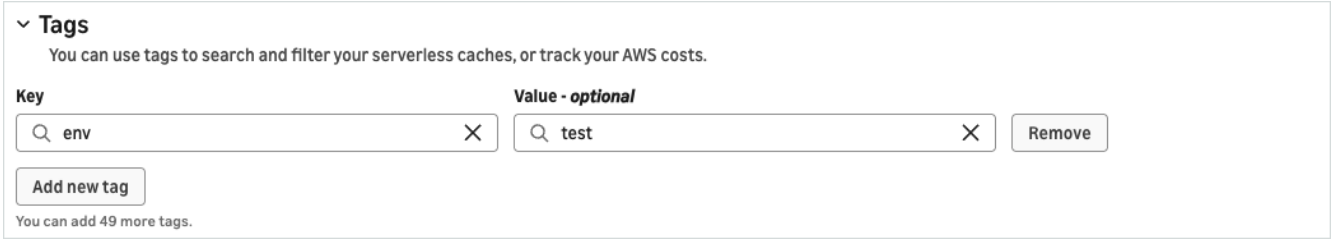

7. For **Connection type**, choose **Public**.



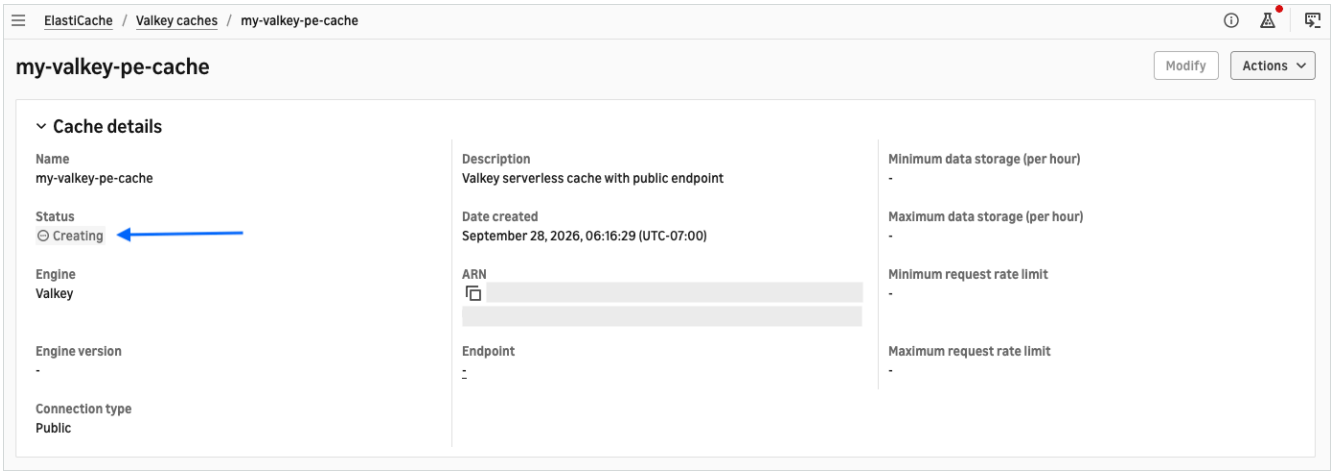

8. By default, ElastiCache uses recommended settings that work for most workloads. Choose **Customize default settings** if you need to change the network type, adjust cache usage limits, or select a different user group. The default user group is `default.iam-user-group`. All users in the group must use IAM authentication.



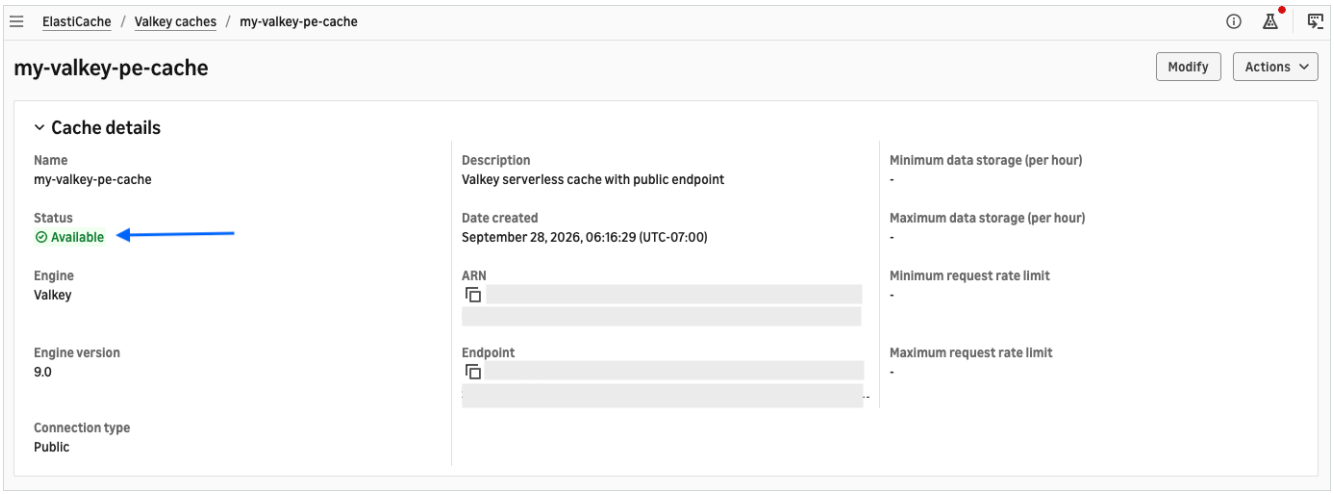

9. (Optional) Under **Tags**, choose **Add new tag** and add tags to search and filter your caches, or track your AWS costs.




10. Choose **Create**. After submission, you are redirected to the Cache details page, where the cache status is shown as **Creating**.




11. Wait until the **Status** on the Cache details page changes from **Creating** to **Available** before connecting to the cache. The page refreshes automatically when the cache is ready.




AWS CLI

Run the following command to create a cache with a public endpoint using the system-managed default IAM user group:

```
aws elasticache create-serverless-cache \
    --serverless-cache-name my-public-cache \
    --engine valkey \
    --major-engine-version 9 \
    --connection-type public \
    --user-group-id default.iam-user-group
```

The command returns a response that includes the cache endpoint:

```
{
    "ServerlessCache": {
        "ServerlessCacheName": "my-public-cache",
        "Status": "creating",
        "Engine": "valkey",
        "MajorEngineVersion": "9",
        "ConnectionType": "public",
        "Endpoint": {
            "Address": "my-public-cache-
x2e9hv.public.serverless.use1.cache.amazonaws.com",
            "Port": 6379
        },
        "UserGroupId": "default.iam-user-group"
    }
}
```

Note the `Endpoint.Address` value. You use this value to connect to your cache.

To use a custom user group instead of the default, replace `default.iam-user-group` with your user group ID. All users in the user group must use IAM authentication.

AWS SDK

The following examples create a cache with a public endpoint.

**Python (boto3)**

```
import boto3

client = boto3.client("elasticache", region_name="us-east-1")

response = client.create_serverless_cache(
    ServerlessCacheName="my-public-cache",
    Engine="valkey",
    MajorEngineVersion="9",
    ConnectionType="public",
    UserGroupId="default.iam-user-group",
)

print(response["ServerlessCache"]["Endpoint"]["Address"])
```

**JavaScript (AWS SDK v3)**

```
import {
  ElastiCacheClient,
  CreateServerlessCacheCommand
} from "@aws-sdk/client-elasticache";

const client = new ElastiCacheClient({ region: "us-east-1" });

const response = await client.send(new CreateServerlessCacheCommand({
  ServerlessCacheName: "my-public-cache",
  Engine: "valkey",
  MajorEngineVersion: "9",
  ConnectionType: "public",
  UserGroupId: "default.iam-user-group",
}));

console.log(response.ServerlessCache?.Endpoint?.Address);
```

**Java (AWS SDK v2)**

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.elasticache.ElastiCacheClient;
import
 software.amazon.awssdk.services.elasticache.model.CreateServerlessCacheRequest;
import
 software.amazon.awssdk.services.elasticache.model.CreateServerlessCacheResponse;

try (ElastiCacheClient client = ElastiCacheClient.builder()
        .region(Region.US_EAST_1)
        .build()) {
    CreateServerlessCacheResponse response = client.createServerlessCache(
        CreateServerlessCacheRequest.builder()
            .serverlessCacheName("my-public-cache")
            .engine("valkey")
            .majorEngineVersion("9")
            .connectionType("public")
            .userGroupId("default.iam-user-group")
            .build());
    System.out.println(response.serverlessCache().endpoint().address());
}
```

After your cache is available, see the section called “Connect to your cache”.

## Connect to a cache with a public endpoint

All public endpoint connections require IAM authentication and TLS 1.3. There are three ways to generate the IAM authentication token:

1. **GLIDE client:** Handles token generation and refresh automatically. No authentication code required.
2. **Developer Toolkit for ElastiCache:** Generates the token for you. Use it with any Valkey or Redis OSS client library.
3. **Manual SigV4 signing:** Generate the token yourself using the AWS SDK.

### Topics


- IAM policy for connecting
- Option 1: GLIDE client with built-in IAM
- Option 2: Developer Toolkit for ElastiCache
- Option 3: Manual SigV4 signing
- Token and session expiration
- Troubleshooting public endpoint connections


### IAM policy for connecting

Before you connect, your IAM user or role needs permission to call `elasticache:Connect`. The AWS managed policy `AdministratorAccess` includes this permission for all resources.

Create an IAM policy that allows the `elasticache:Connect` action on the ElastiCache cache and ElastiCache user:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticache:Connect",
            "Resource": [
                "arn:aws:elasticache:us-east-1:111122223333:serverlesscache:my-public-
cache",
                "arn:aws:elasticache:us-east-1:111122223333:user:default.iam-user"
```

```
            ]
        }
    ]
}
```

### Option 1: GLIDE client with built-in IAM

GLIDE 2.2 and later include built-in IAM authentication support. You configure your AWS credentials, and GLIDE handles token generation, caching, and refresh automatically. For installation instructions, see the GLIDE documentation on the GLIDE website.

#### Python

```
from glide import (
    GlideClusterClient, GlideClusterClientConfiguration, NodeAddress,
    ServerCredentials, IamAuthConfig, ServiceType,
)

config = GlideClusterClientConfiguration(
    addresses=[NodeAddress("my-public-cache-
x2e9hv.public.serverless.use1.cache.amazonaws.com", 6379)],
    credentials=ServerCredentials(
        username="default.iam-user",
        iam_config=IamAuthConfig(
            cluster_name="my-public-cache",
            service=ServiceType.ELASTICACHE,
            region="us-east-1",
        ),
    ),
    use_tls=True,
)

client = await GlideClusterClient.create(config)
await client.set("key", "value")
result = await client.get("key")
print(result)  # b"value"
```

#### Java

```
import glide.api.GlideClusterClient;
import glide.api.models.configuration.IamAuthConfig;
import glide.api.models.configuration.ServiceType;
```

```
import glide.api.models.configuration.ServerCredentials;
import glide.api.models.configuration.GlideClusterClientConfiguration;
import glide.api.models.configuration.NodeAddress;

import java.util.List;
import java.util.Collections;

List<NodeAddress> nodes = Collections.singletonList(
    NodeAddress.builder()
        .host("my-public-cache-x2e9hv.public.serverless.use1.cache.amazonaws.com")
        .port(6379)
        .build()
);

IamAuthConfig iamConfig = IamAuthConfig.builder()
    .clusterName("my-public-cache")
    .service(ServiceType.ELASTICACHE)
    .region("us-east-1")
    .build();

GlideClusterClientConfiguration config = GlideClusterClientConfiguration.builder()
    .addresses(nodes)
    .credentials(ServerCredentials.builder()
        .username("default.iam-user")
        .iamConfig(iamConfig)
        .build())
    .useTLS(true)
    .build();

GlideClusterClient client = GlideClusterClient.createClient(config).get();
client.set("key", "value").get();
String result = client.get("key").get();
System.out.println(result);  // "value"
```

### Option 2: Developer Toolkit for ElastiCache

The Developer Toolkit for ElastiCache (`developer-toolkit-elasticache`) is a standalone Python library and CLI that generates IAM authentication tokens. Use it with a Python Valkey or Redis OSS client library, such as valkey-py.

### Install

```
pip install developer-toolkit-elasticache
```

### Python with valkey-py

```
import valkey
from valkey.credentials import CredentialProvider

from developer_toolkit_elasticache import ElastiCacheIAMAuthTokenProvider

class ElastiCacheCredentialProvider(CredentialProvider):
    def __init__(self, auth):
        self._auth = auth

    def get_credentials(self):
        return self._auth.user_id, self._auth.get_token()

auth = ElastiCacheIAMAuthTokenProvider(
    serverless_cache_name="my-public-cache",
    user_id="default.iam-user",
    region="us-east-1",
)

client = valkey.ValkeyCluster(
    host="my-public-cache-x2e9hv.public.serverless.use1.cache.amazonaws.com",
    port=6379,
    ssl=True,
    credential_provider=ElastiCacheCredentialProvider(auth),
)

client.set("key", "value")
result = client.get("key")
print(result)  # b"value"
```

The Developer Toolkit does not automatically refresh tokens. IAM authentication tokens are valid for 15 minutes, so your application must generate a new token before the current token expires. The credentials provider in the preceding example calls `get_token()` for each new connection.

### CLI with the Developer Toolkit

To connect with valkey-cli or redis-cli, use the Developer Toolkit CLI to generate a token:

```
export VALKEYCLI_AUTH=$(developer-toolkit-elasticache generate_iam_auth_token \
    --serverless-cache-name my-public-cache \
    --user-id default.iam-user \
    --region us-east-1)
```

```
valkey-cli -c -h my-public-cache-x2e9hv.public.serverless.use1.cache.amazonaws.com \
    -p 6379 \
    --tls \
    --user default.iam-user
```

VALKEYCLI_AUTH requires valkey-cli 9.0 or later. For redis-cli, use REDISCLI_AUTH instead of VALKEYCLI_AUTH.

For source code and additional examples, see the Developer Toolkit for ElastiCache repository on GitHub.

**Option 3: Manual SigV4 signing**

Generate the IAM authentication token manually using the AWS SDK. For signing code examples, see the section called “Connect with token as password”.

**Token and session expiration**

- **Token expiration:** IAM authentication tokens are valid for 15 minutes. GLIDE refreshes the token automatically. If you use the Developer Toolkit or manual signing, generate a new token for each new connection. Token expiration does not affect established connections. If you sign the token with temporary credentials, the token expires when those credentials expire, if that is sooner than 15 minutes.
- **Session expiration:** Connections expire after 12 hours and reconnect automatically. Revoking an IAM principal's `elasticache:Connect` permission does not immediately disconnect active sessions. To disconnect a user immediately, remove the user from the cache's user group.

**Troubleshooting public endpoint connections**

The following table lists common connection errors, their causes, and how to resolve them.

| Error | Cause | Resolution |
| --- | --- | --- |
| `WRONGPASS` or authentication failure | Expired IAM token, incorrect username, missing IAM permissions, cache name not in | Verify that the token is fresh (less than 15 minutes old), the username matches a user in the cache's user group, the IAM principal has `elasticache:Connect` permission on both the cache ARN and the user ARN, |

| Error | Cause | Resolution |
| --- | --- | --- |
| | lowercase, or expired temporary credentia ls | the cache name in your token signing code is lowercase, and, if you signed the token with temporary credentials, that those credentials had not expired. |
| `NOAUTH Authentic ation required` | The client sent a command before authenticating. A public endpoint connection must authenticate with `AUTH` or `HELLO ... AUTH` as its first command. Some client libraries send a command such as `CLIENT SETINFO` or `PING` on connect, before credentia ls are applied. The connection is closed after repeated unauthenticated commands. | Configure your client so that authentication happens as part of connection setup — pass the token as the connection password or through a credential provider, rather than calling `AUTH` after connecting. GLIDE does this automatically. |

| Error | Cause | Resolution |
| --- | --- | --- |
| `ERR service is not available` | This error is returned for any new connection that fails before authentication completes. To avoid disclosing engine-internal state to an unauthenticated caller, ElastiCache reports this same generic message for several distinct underlying causes: IAM authentication is temporarily unavailable, you are being rate-limited due to IAM authentication throttling, or the engine is temporarily overloaded (`BUSY`, `CLUSTERDOWN` , or max clients reached). On an already-authenticated connection that re-runs `AUTH` or `HELLO`, ElastiCache returns a specific error instead, such as `ERR IAM Authentication service is not available` or `ERR Exceeded limit` | If you have recently opened a high rate of new connections or re-authentications, reduce that rate and retry with exponential backoff. If your connection and re-authentication rate is normal, check the AWS Health Dashboard for ElastiCache service issues. |

| Error | Cause | Resolution |
| --- | --- | --- |
| | `of IAM Authentic ation requests .` | |
| Connection timeout | Network connectiv ity issue, incorrect endpoint, or TLS not enabled | Verify that the endpoint address is correct, TLS is enabled in your client configuration, and your network allows outbound connections to port 6379. |
| `TLS handshake failure` | The client does not support TLS 1.3. Caches with a public endpoint require TLS 1.3. | Update your client or runtime to a version that supports TLS 1.3. |

## Create a Valkey serverless cache in a VPC

### AWS Management Console

To create a new cache using the ElastiCache console:

1. Sign in to the AWS Management Console and open the https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Valkey caches**.
3. On the right side of the console, choose **Create Valkey cache**
4. In the **Cache settings** enter a **Name**. You can optionally enter a **description** for the cache.
5. Leave the default settings selected.
6. Click **Create** to create the cache.
7. Once the cache is in "ACTIVE" status, you can begin writing and reading data to the cache. .

### AWS CLI

The following AWS CLI example creates a new cache using create-serverless-cache.

### Linux

```
aws elasticache create-serverless-cache \
    --serverless-cache-name CacheName \
    --engine valkey
```

**Windows**

```
aws elasticache create-serverless-cache ^
    --serverless-cache-name CacheName ^
    --engine valkey
```

Note that the value of the Status field is set to `CREATING`.

To verify that ElastiCache has finished creating the cache, use the `describe-serverless-caches` command.

**Linux**

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

**Windows**

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

After creating the new cache, proceed to Read and write data to the cache.

## Read and write data to the cache

This section assumes that you've created an Amazon EC2 instance and can connect to it. For instructions on how to do this, see the Amazon EC2 Getting Started Guide.

This section also assumes that you have setup VPC access and security group settings for the EC2 instance from where you are connecting to your cache, and setup valkey-cli on your EC2 instance. For more information on that step see Setting up ElastiCache.

In addition to the steps below, if you have a large or global application you can greatly increase read performance by reading from replicas. For more information on this more advanced step see Best Practices for using Read Replicas.

**Find your cache endpoint**

### AWS Management Console

To find your cache's endpoint using the ElastiCache console:

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Valkey caches**.
3. On the right side of the console, click on the name of the cache that you just created.
4. In the **Cache details**, locate and copy the cache endpoint.

### AWS CLI

The following AWS CLI example shows to find the endpoint for your new cache using the describe-serverless-caches command. Once you have run the command, look for the "Endpoint" field.

#### Linux

```
aws elasticache describe-serverless-caches \
  --serverless-cache-name CacheName
```

#### Windows

```
aws elasticache describe-serverless-caches ^
  --serverless-cache-name CacheName
```

### Connect to your Valkey Cache (Linux)

Now that you have the endpoint you need, you can log in to your EC2 instance and connect to the cache. In the following example, you use the *valkey-cli* utility to connect to a cluster. The following command connects to a cache (note: replace cache-endpoint with the endpoint you retrieved in the previous step).

```
src/valkey-cli -h cache-endpoint --tls -p 6379
set a "hello"          // Set key "a" with a string value and no expiration
OK
get a                  // Get value for key "a"
"hello"
```

### Connect to your Valkey Cache (Windows)

Now that you have the endpoint you need, you can log in to your EC2 instance and connect to the cache. In the following example, you use the *valkey-cli* utility to connect to a cluster. The following command connects to a cache. Open the Command Prompt and change to the Valkey or Redis OSS directory and run the command (note: replace Cache_Endpoint with the endpoint you retrieved in the previous step).

```
c:\Valkey>valkey-cli -h Valkey_Cluster_Endpoint --tls -p 6379
set a "hello"          // Set key "a" with a string value and no expiration
OK
get a                  // Get value for key "a"
"hello"
```

You may now proceed to (Optional) Clean up.

## (Optional) Clean up

If you no longer need the Amazon ElastiCache cache that you created, you can delete it. This step helps ensure that you are not charged for resources that you are not using. You can use the ElastiCache console, the AWS CLI, or the ElastiCache API to delete your cache.

### AWS Management Console

To delete your cache using the console:

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Valkey Caches**.
3. Choose the radio button next to the cache you want to delete.
4. Select **Actions** on the top right, and select **Delete**.
5. You can optionally choose to take a final snapshot before you delete your cache.
6. In the **Delete** confirmation screen, re-enter the cache name and choose **Delete** to delete the cluster, or choose **Cancel** to keep the cluster.

As soon as your cache moves in to the **DELETING** status, you stop incurring charges for it.

### AWS CLI

The following AWS CLI example deletes a cache using the delete-serverless-cache command.

**Linux**

```
aws elasticache delete-serverless-cache \
  --serverless-cache-name CacheName
```

**Windows**

```
aws elasticache delete-serverless-cache ^
  --serverless-cache-name CacheName
```

Note that the value of the **Status** field is set to **DELETING**.

You may now proceed to Next Steps.

### Next Steps

For more information about ElastiCache see the following pages:

- Working with ElastiCache
- Scaling ElastiCache
- Logging and monitoring in Amazon ElastiCache
- ElastiCache best practices and caching strategies
- Snapshot and restore
- Amazon SNS monitoring of ElastiCache events

## Create a Redis OSS serverless cache

In this step, you create a new cache in Amazon ElastiCache.

**AWS Management Console**

To create a new cache using the ElastiCache console:

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. In the navigation pane on the left side of the console, choose **Redis OSS caches**.
3. On the right side of the console, choose **Create Redis OSS cache**
4. In the **Cache settings** enter a **Name**. You can optionally enter a **description** for the cache.
5. Leave the default settings selected.
6. Click **Create** to create the cache.
7. Once the cache is in "ACTIVE" status, you can begin writing and reading data to the cache. .

### AWS CLI

The following AWS CLI example creates a new cache using create-serverless-cache.

#### Linux

```
aws elasticache create-serverless-cache \
    --serverless-cache-name CacheName \
    --engine redis
```

#### Windows

```
aws elasticache create-serverless-cache ^
    --serverless-cache-name CacheName ^
    --engine redis
```

Note that the value of the Status field is set to `CREATING`.

To verify that ElastiCache has finished creating the cache, use the `describe-serverless-caches` command.

#### Linux

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

#### Windows

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

After creating the new cache, proceed to Read and write data to the cache.

# Read and write data to the cache

This section assumes that you've created an Amazon EC2 instance and can connect to it. For instructions on how to do this, see the Amazon EC2 Getting Started Guide.

This section also assumes that you have setup VPC access and security group settings for the EC2 instance from where you are connecting to your cache, and setup valkey-cli on your EC2 instance. For more information on that step see Setting up ElastiCache.

### Find your cache endpoint

#### AWS Management Console

To find your cache's endpoint using the ElastiCache console:

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Valkey caches Redis OSS caches**.
3. On the right side of the console, click on the name of the cache that you just created.
4. In the **Cache details**, locate and copy the cache endpoint.

#### AWS CLI

The following AWS CLI example shows to find the endpoint for your new cache using the describe-serverless-caches command. Once you have run the command, look for the "Endpoint" field.

**Linux**

```
aws elasticache describe-serverless-caches \
  --serverless-cache-name CacheName
```

**Windows**

```
aws elasticache describe-serverless-caches ^
  --serverless-cache-name CacheName
```

### Connect to your Valkey or Redis OSS Cache (Linux)

Now that you have the endpoint you need, you can log in to your EC2 instance and connect to the cache. In the following example, you use the *valkey-cli* utility to connect to a cluster. The following

command connects to a cache (note: replace cache-endpoint with the endpoint you retrieved in the previous step).

```
src/valkey-cli -h cache-endpoint --tls -p 6379
set a "hello"          // Set key "a" with a string value and no expiration
OK
get a                  // Get value for key "a"
"hello"
```

**Connect to your Valkey or Redis OSS Cache (Windows)**

Now that you have the endpoint you need, you can log in to your EC2 instance and connect to the cache. In the following example, you use the *valkey-cli* utility to connect to a cluster. The following command connects to a cache. Open the Command Prompt and change to the Valkey directory and run the command (note: replace Cache_Endpoint with the endpoint you retrieved in the previous step).

```
c:\Redis>valkey-cli -h Redis_Cluster_Endpoint --tls -p 6379
set a "hello"          // Set key "a" with a string value and no expiration
OK
get a                  // Get value for key "a"
"hello"
```

You may now proceed to (Optional) Clean up.

## (Optional) Clean up

If you no longer need the Amazon ElastiCache cache that you created, you can delete it. This step helps ensure that you are not charged for resources that you are not using. You can use the ElastiCache console, the AWS CLI, or the ElastiCache API to delete your cache.

**AWS Management Console**

To delete your cache using the console:

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Valkey or Redis OSS Caches**.
3. Choose the radio button next to the cache you want to delete.
4. Select **Actions** on the top right, and select **Delete**.

5. You can optionally choose to take a final snapshot before you delete your cache.
6. In the **Delete** confirmation screen, re-enter the cache name and choose **Delete** to delete the cluster, or choose **Cancel** to keep the cluster.

As soon as your cache moves in to the **DELETING** status, you stop incurring charges for it.

**AWS CLI**

The following AWS CLI example deletes a cache using the delete-serverless-cache command.

**Linux**

```
aws elasticache delete-serverless-cache \
  --serverless-cache-name CacheName
```

**Windows**

```
aws elasticache delete-serverless-cache ^
  --serverless-cache-name CacheName
```

Note that the value of the **Status** field is set to **DELETING**.

You may now proceed to Next Steps.

## Next Steps

For more information about ElastiCache see the following pages:

- Working with ElastiCache
- Scaling ElastiCache
- Logging and monitoring in Amazon ElastiCache
- ElastiCache best practices and caching strategies
- Snapshot and restore
- Amazon SNS monitoring of ElastiCache events

# Create a Memcached serverless cache

**AWS Management Console**

To create a new Memcached serverless cache using the ElastiCache console:

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Memcached Caches**.
3. On the right side of the console, choose **Create Memcached cache**.
4. In the **Cache settings** enter a **Name**. You can optionally enter a **description** for the cache.
5. Leave the default settings selected.
6. Click **Create** to create the cache.
7. Once the cache is in "ACTIVE" status, you can begin writing and reading data to the cache.

To create a new cache using the AWS CLI

The following AWS CLI example creates a new cache using create-serverless-cache.

**Linux**

```
aws elasticache create-serverless-cache \
  --serverless-cache-name CacheName \
  --engine memcached
```

**Windows**

```
aws elasticache create-serverless-cache ^
  --serverless-cache-name CacheName ^
  --engine memcached
```

Note that the value of the Status field is set to `CREATING`.

To verify that ElastiCache has finished creating the cache, use the `describe-serverless-caches` command.

**Linux**

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

**Windows**

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

After creating the new cache, proceed to Read and write data to the cache.

## Read and write data to the cache

This section assumes that you've created an Amazon EC2 instance and can connect to it. For instructions on how to do this, see the Amazon EC2 Getting Started Guide.

By default, ElastiCache creates a cache in your default VPC. Make sure that your EC2 instance is also created in the default VPC, so that it is able to connect to the cache.

### Find your cache endpoint

#### AWS Management Console

To find your cache's endpoint using the ElastiCache console:

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane on the left side of the console, choose **Memcached Caches**.
3. On the right side of the console, click on the name of the cache that you just created.
4. In the **Cache details**, locate and copy the cache endpoint.

#### AWS CLI

The following AWS CLI example shows to find the endpoint for your new cache using the describe-serverless-caches command. Once you have run the command, look for the "Endpoint" field.

**Linux**

```
aws elasticache describe-serverless-caches \
  --serverless-cache-name CacheName
```

**Windows**

```
aws elasticache describe-serverless-caches ^
  --serverless-cache-name CacheName
```

### Connect using OpenSSL

For information on how to connect using OpenSSL, see ElastiCache in-transit encryption (TLS)

### Connect using Memcached Java client

> **Note**
>
> The following example uses the AWS fork of the spymemcached client, aws-elasticache-cluster-client-memcached-for-java. The `setSSLContext` method is only available in this fork and is not part of the standard open-source spymemcached library.

```
import java.security.KeyStore;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;
import net.spy.memcached.AddrUtil;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.FailureMode;
import net.spy.memcached.MemcachedClient;

public class TLSDemo {
    public static void main(String[] args) throws Exception {
        ConnectionFactoryBuilder connectionFactoryBuilder = new
 ConnectionFactoryBuilder();
        // Build SSLContext
        TrustManagerFactory tmf =
 TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init((KeyStore) null);
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, tmf.getTrustManagers(), null);
        // Create the client in TLS mode
        connectionFactoryBuilder.setSSLContext(sslContext);
        // Set Failure Mode to Retry
        connectionFactoryBuilder.setFailureMode(FailureMode.Retry);
        MemcachedClient client = new MemcachedClient(connectionFactoryBuilder.build(),
 AddrUtil.getAddresses("mycluster-fnjyzo.serverless.use1.cache.amazonaws.com:11211"));

        // Store a data item for an hour.
        client.set("theKey", 3600, "This is the data value");
    }
}
```

### Connect using Memcached PHP client

```
<?php
$cluster_endpoint = "mycluster.serverless.use1.cache.amazonaws.com";
$server_port = 11211;

/* Initialize a persistent Memcached client in TLS mode */
$tls_client = new Memcached('persistent-id');
$tls_client->addServer($cluster_endpoint, $server_port);
if(!$tls_client->setOption(Memcached::OPT_USE_TLS, 1)) {
    echo $tls_client->getLastErrorMessage(), "\n";
    exit(1);
}
$tls_config = new MemcachedTLSContextConfig();
$tls_config->hostname = '*.serverless.use1.cache.amazonaws.com';
$tls_config->skip_cert_verify = false;
$tls_config->skip_hostname_verify = false;
$tls_client->createAndSetTLSContext((array)$tls_config);

 /* store the data for 60 seconds in the cluster */
$tls_client->set('key', 'value', 60);
?>
```

### Connect using Memcached Python client (Pymemcache)

See https://pymemcache.readthedocs.io/en/latest/getting_started.html

```
import ssl
from pymemcache.client.base import Client

context = ssl.create_default_context()
cluster_endpoint = <To be taken from the AWS CLI / console>
target_port = 11211
memcached_client = Client((cluster_endpoint, target_port), tls_context=context)
memcached_client.set("key", "value", expire=500, noreply=False)
assert self.memcached_client.get("key").decode() == "value"
```

### Connect using Memcached NodeJS/TS client (Electrode-IO memcache)

See https://github.com/electrode-io/memcache and https://www.npmjs.com/package/memcache-client

Install via `npm i memcache-client`

In the application, create a memcached TLS client as follows:

```
var memcache = require("memcache-client");
const client = new memcache.MemcacheClient({server: "{cluster_endpoint}:11211", tls:
 {}});
client.set("key", "value");
```

### Connect using Memcached Rust client (rust-memcache)

See https://crates.io/crates/memcache and https://github.com/aisk/rust-memcache.

```
// create connection with to memcached server node:
let client = memcache::connect("memcache+tls://<cluster_endpoint>:11211?
verify_mode=none").unwrap();

// set a string value
client.set("foo", "bar", 0).unwrap();
```

### Connect using Memcached Go client (Gomemcache)

See https://github.com/bradfitz/gomemcache

```
c := New(net.JoinHostPort("{cluster_endpoint}", strconv.Itoa(port)))
c.DialContext = func(ctx context.Context, network, addr string) (net.Conn, error) {
var td tls.Dialer
td.Config = &tls.Config{}
return td.DialContext(ctx, network, addr)
}
foo := &Item{Key: "foo", Value: []byte("fooval"), Flags: 123}
err := c.Set(foo)
```

### Connect using Memcached Ruby client (Dalli)

See https://github.com/petergoldstein/dalli

```
require 'dalli'
ssl_context = OpenSSL::SSL::SSLContext.new
ssl_context.ssl_version = :SSLv23
ssl_context.verify_hostname = true
ssl_context.verify_mode = OpenSSL::SSL::VERIFY_PEER
client = Dalli::Client.new("<cluster_endpoint>:11211", :ssl_context => ssl_context);
```

```
client.get("abc")
```

### Connect using Memcached .NET client (EnyimMemcachedCore)

See https://github.com/cnblogs/EnyimMemcachedCore

```
"MemcachedClient": {
"Servers": [
{
"Address": "{cluster_endpoint}",
"Port": 11211
}
],
"UseSslStream":  true
}
```

You may now proceed to (Optional) Clean up.

# (Optional) Clean up

### Using the AWS Management Console

The following procedure deletes a single cache from your deployment. To delete multiple caches, repeat the procedure for each cache that you want to delete. You do not need to wait for one cache to finish deleting before starting the procedure to delete another cache.

### To delete a cache

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the ElastiCache console dashboard, choose the engine that is running on the cache that you want to delete. A list of all caches running that engine appears.
3. To choose the cache to delete, choose the cache's name from the list of caches.

   > **Important**
   >
   > You can only delete one cache at a time from the ElastiCache console. Choosing multiple caches disables the delete operation.

4. For **Actions**, choose **Delete**.
5. In the **Delete Cache** confirmation screen, choose **Delete** to delete the cache, or choose **Cancel** to keep the cluster.
6. If you chose **Delete**, the status of the cache changes to *deleting*.

As soon as your cache moves in to the **DELETING** status, you stop incurring charges for it.

### Using the AWS CLI

The following code deletes the cache my-cache.

```
aws elasticache delete-serverless-cache --serverless-cache-name my-cache
```

The delete-serverless-cache CLI action only deletes one serverless cache. To delete multiple caches, call delete-serverless-cache for each serverless cache that you want to delete. You do not need to wait for one serverless cache to finish deleting before deleting another.

**For Linux, macOS, or Unix:**

```
aws elasticache delete-serverless-cache \
  --serverless-cache-name my-cache
```

**For Windows:**

```
aws elasticache delete-serverless-cache ^
  --serverless-cache-name my-cache
```

For more information, see the AWS CLI for ElastiCache topic delete-serverless-cache.

You may now proceed to Next Steps.

### Next Steps

For more information about ElastiCache see:

- Working with ElastiCache
- Scaling ElastiCache
- Quotas for ElastiCache
- ElastiCache best practices and caching strategies
- Viewing ElastiCache events

## Choosing a network type for serverless caches

ElastiCache serverless caches support the Internet Protocol versions 4 and 6 (IPv4 and IPv6). When creating a serverless cache, you choose one of the following network types:

- **IPv4** – The cache accepts only IPv4 connections.
- **IPv6** – The cache accepts only IPv6 connections.
- **Dual stack** – The cache accepts both IPv4 and IPv6 connections.

For caches with a public endpoint, the network type defaults to IPv4. You can select a different network type when creating the cache. Subnet configuration is not required.

For dual stack serverless caches, the IP protocol used for a connection depends on how your client resolves the cache endpoint's DNS hostname.

There are no additional charges for accessing ElastiCache over IPv6.

> **Note**
>
> The network type can only be set when creating a serverless cache. You cannot change the network type after the cache is created.

## Configuring subnets for network type

This section applies to caches with a VPC connection type. Caches with a public endpoint do not require subnet configuration.

When you create a serverless cache, you can provide subnet IDs. ElastiCache uses those subnets to allocate IP addresses for your cache. The subnets you provide must support the network type you choose:

- **IPv4** – Subnets must have IPv4 address space. Dual stack subnets (with both IPv4 and IPv6) are also supported.
- **IPv6** – Subnets must be IPv6-only. Dual stack subnets are not supported.
- **Dual stack** – Subnets must have both IPv4 and IPv6 address space.

If you do not specify a network type, ElastiCache defaults to IPv4 unless all provided subnets are IPv6-only, in which case it defaults to IPv6. If you do not provide subnet IDs, ElastiCache selects default subnets in your VPC.

## Using the AWS Management Console

When creating a serverless cache using the console, choose **Customize default settings** under **Default settings**. In the **Connectivity** section, choose a **Network type**: **IPv4**, **IPv6**, or **Dual stack**. IPv4 is selected by default.

## Using the AWS CLI

When creating a serverless cache using the AWS CLI, use the `--network-type` parameter with the **create-serverless-cache** command.

For Linux, macOS, or Unix:

```
aws elasticache create-serverless-cache \
  --serverless-cache-name <cache-name> \
  --engine <engine> \
  --network-type <network-type> \
  --subnet-ids <subnet-id-1> <subnet-id-2>
```

For Windows:

```
aws elasticache create-serverless-cache ^
  --serverless-cache-name <cache-name> ^
  --engine <engine> ^
  --network-type <network-type> ^
  --subnet-ids <subnet-id-1> <subnet-id-2>
```

Replace:

- *<cache-name>* – A name for your serverless cache.
- *<engine>* – The engine for your cache: `valkey`, `redis`, or `memcached`.
- *<network-type>* – The network type: `ipv4`, `ipv6`, or `dual_stack`.
- *<subnet-id-1>*, *<subnet-id-2>* – The IDs of the subnets for your cache. The subnets must support the chosen network type.

For more information about the **create-serverless-cache** command, see create-serverless-cache in the AWS CLI reference.

For information about choosing a network type for node-based clusters, see the section called "Choosing a network type in ElastiCache".

# Tutorials: Getting started with Python and ElastiCache

This section contains hands-on tutorials to help you learn about ElastiCache for Valkey and Redis OSS. We encourage you to work through one of the language-specific tutorials.

> **Note**
>
> AWS SDKs are available for a wide variety of languages. For a complete list, see Tools for Amazon Web Services.

**Topics**


- Python and ElastiCache


# Python and ElastiCache

In this tutorial, you use the AWS SDK for Python (Boto3) to write simple programs to perform the following ElastiCache operations:

- Create ElastiCache for Redis OSS clusters (cluster mode enabled and cluster mode disabled)
- Check if users or user groups exist, otherwise create them. (This feature is availably with Valkey 7.2 and onwards, and with Redis OSS 6.0 to 7.1.)
- Connect to ElastiCache
- Perform operations such as setting and getting strings, reading from and writing to steams and publishing and subscribing from Pub/Sub channel.

As you work through this tutorial, you can refer to the AWS SDK for Python (Boto) documentation. The following section is specific to ElastiCache: ElastiCache low-level client

## Tutorial Prerequisites

- Set up an AWS access key to use the AWS SDKs. For more information, see Setting up ElastiCache.
- Install Python 3.0 or later. For more information, see https://www.python.org/downloads. For instructions, see Quickstart in the Boto 3 documentation.

**Topics**


- Tutorial: Creating ElastiCache clusters and users
- Tutorial: Connecting to ElastiCache
- Usage examples


## Tutorial: Creating ElastiCache clusters and users

The following examples use the boto3 SDK for ElastiCache for Redis OSS management operations (cluster or user creation) and redis-py/redis-py-cluster for data handling.

**Topics**


- Create a cluster mode disabled cluster
- Create a cluster mode disabled cluster with TLS and RBAC
- Create a cluster mode enabled cluster
- Create a cluster mode enabled cluster with TLS and RBAC
- Check if users/usergroup exists, otherwise create them


### Create a cluster mode disabled cluster

Copy the following program and paste it into a file named *CreateClusterModeDisabledCluster.py*.

```
import boto3
import logging

logging.basicConfig(level=logging.INFO)
client = boto3.client('elasticache')

def
 create_cluster_mode_disabled(CacheNodeType='cache.t3.small',EngineVersion='6.0',NumCacheCluste
 cluster',ReplicationGroupId=None):
    """Creates an ElastiCache Cluster with cluster mode disabled

    Returns a dictionary with the API response

    :param CacheNodeType: Node type used on the cluster. If not specified,
 cache.t3.small will be used
    Refer to https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/
CacheNodes.SupportedTypes.html for supported node types
    :param EngineVersion: Engine version to be used. If not specified, latest will be
 used.
    :param NumCacheClusters: Number of nodes in the cluster. Minimum 1 (just a primary
 node) and maximun 6 (1 primary and 5 replicas).
    If not specified, cluster will be created with 1 primary and 1 replica.
    :param ReplicationGroupDescription: Description for the cluster.
    :param ReplicationGroupId: Name for the cluster
    :return: dictionary with the API results

    """
    if not ReplicationGroupId:
        return 'ReplicationGroupId parameter is required'
```

```
    response = client.create_replication_group(
        AutomaticFailoverEnabled=True,
        CacheNodeType=CacheNodeType,
        Engine='valkey',
        EngineVersion=EngineVersion,
        NumCacheClusters=NumCacheClusters,
        ReplicationGroupDescription=ReplicationGroupDescription,
        ReplicationGroupId=ReplicationGroupId,
        SnapshotRetentionLimit=30,
    )
    return response


if __name__ == '__main__':

    # Creates an ElastiCache Cluster mode disabled cluster, based on cache.m6g.large
 nodes, Valkey 8.0, one primary and two replicas
    elasticacheResponse = create_cluster_mode_disabled(
        #CacheNodeType='cache.m6g.large',
        EngineVersion='8.0',
        NumCacheClusters=3,
        ReplicationGroupDescription='Valkey cluster mode disabled with replicas',
        ReplicationGroupId='valkey202104053'
        )

    logging.info(elasticacheResponse)
```

To run the program, enter the following command:

```
python CreateClusterModeDisabledCluster.py
```

For more information, see Managing clusters in ElastiCache.

**Create a cluster mode disabled cluster with TLS and RBAC**

To ensure security, you can use Transport Layer Security (TLS) and Role-Based Access Control (RBAC) when creating a cluster mode disabled cluster. Unlike Valkey or Redis OSS AUTH, where all authenticated clients have full replication group access if their token is authenticated, RBAC enables you to control cluster access through user groups. These user groups are designed as a way to organize access to replication groups. For more information, see Role-Based Access Control (RBAC).

Copy the following program and paste it into a file named *ClusterModeDisabledWithRBAC.py*.

```
import boto3
import logging

logging.basicConfig(level=logging.INFO)
client = boto3.client('elasticache')

def
 create_cluster_mode_disabled_rbac(CacheNodeType='cache.t3.small',EngineVersion='6.0',NumCacheC
 cluster',ReplicationGroupId=None, UserGroupIds=None,
 SecurityGroupIds=None,CacheSubnetGroupName=None):
    """Creates an ElastiCache Cluster with cluster mode disabled and RBAC

    Returns a dictionary with the API response

    :param CacheNodeType: Node type used on the cluster. If not specified,
 cache.t3.small will be used
    Refer to https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/
CacheNodes.SupportedTypes.html for supported node types
    :param EngineVersion: Engine version to be used. If not specified, latest will be
 used.
    :param NumCacheClusters: Number of nodes in the cluster. Minimum 1 (just a primary
 node) and maximun 6 (1 primary and 5 replicas).
    If not specified, cluster will be created with 1 primary and 1 replica.
    :param ReplicationGroupDescription: Description for the cluster.
    :param ReplicationGroupId: Mandatory name for the cluster.
    :param UserGroupIds: The ID of the user group to be assigned to the cluster.
    :param SecurityGroupIds: List of security groups to be assigned. If not defined,
 default will be used
    :param CacheSubnetGroupName: subnet group where the cluster will be placed. If not
 defined, default will be used.
    :return: dictionary with the API results

    """
    if not ReplicationGroupId:
        return {'Error': 'ReplicationGroupId parameter is required'}
    elif not isinstance(UserGroupIds,(list)):
        return {'Error': 'UserGroupIds parameter is required and must be a list'}

    params={'AutomaticFailoverEnabled': True,
            'CacheNodeType': CacheNodeType,
            'Engine': 'valkey',
            'EngineVersion': EngineVersion,
```

```
            'NumCacheClusters': NumCacheClusters,
            'ReplicationGroupDescription': ReplicationGroupDescription,
            'ReplicationGroupId': ReplicationGroupId,
            'SnapshotRetentionLimit': 30,
            'TransitEncryptionEnabled': True,
            'UserGroupIds':UserGroupIds
        }

    # defaults will be used if CacheSubnetGroupName or SecurityGroups are not explicit.
    if isinstance(SecurityGroupIds,(list)):
        params.update({'SecurityGroupIds':SecurityGroupIds})
    if CacheSubnetGroupName:
        params.update({'CacheSubnetGroupName':CacheSubnetGroupName})

    response = client.create_replication_group(**params)
    return response

if __name__ == '__main__':

    # Creates an ElastiCache Cluster mode disabled cluster, based on cache.m6g.large
 nodes, Valkey 8.0, one primary and two replicas.
    # Assigns the existent user group "mygroup" for RBAC authentication

    response=create_cluster_mode_disabled_rbac(
        CacheNodeType='cache.m6g.large',
        EngineVersion='8.0',
        NumCacheClusters=3,
        ReplicationGroupDescription='Valkey cluster mode disabled with replicas',
        ReplicationGroupId='valkey202104',
        UserGroupIds=[
            'mygroup'
        ],
        SecurityGroupIds=[
            'sg-7cc73803'
        ],
        CacheSubnetGroupName='default'
    )

    logging.info(response)
```

To run the program, enter the following command:

```
python ClusterModeDisabledWithRBAC.py
```

For more information, see Managing clusters in ElastiCache.

### Create a cluster mode enabled cluster

Copy the following program and paste it into a file named *ClusterModeEnabled.py*.

```
import boto3
import logging

logging.basicConfig(level=logging.INFO)
client = boto3.client('elasticache')

def
 create_cluster_mode_enabled(CacheNodeType='cache.t3.small',EngineVersion='6.0',NumNodeGroups=1
 ReplicationGroupDescription='Sample cache with cluster mode
 enabled',ReplicationGroupId=None):
    """Creates an ElastiCache Cluster with cluster mode enabled

    Returns a dictionary with the API response

    :param CacheNodeType: Node type used on the cluster. If not specified,
 cache.t3.small will be used
    Refer to https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/
CacheNodes.SupportedTypes.html for supported node types
    :param EngineVersion: Engine version to be used. If not specified, latest will be
 used.
    :param NumNodeGroups: Number of shards in the cluster. Minimum 1 and maximun 90.
    If not specified, cluster will be created with 1 shard.
    :param ReplicasPerNodeGroup: Number of replicas per shard. If not specified 1
 replica per shard will be created.
    :param ReplicationGroupDescription: Description for the cluster.
    :param ReplicationGroupId: Name for the cluster
    :return: dictionary with the API results

    """
    if not ReplicationGroupId:
        return 'ReplicationGroupId parameter is required'

    response = client.create_replication_group(
        AutomaticFailoverEnabled=True,
        CacheNodeType=CacheNodeType,
        Engine='valkey',
        EngineVersion=EngineVersion,
        ReplicationGroupDescription=ReplicationGroupDescription,
```

```
        ReplicationGroupId=ReplicationGroupId,
    #   Creates a cluster mode enabled cluster with 1 shard(NumNodeGroups), 1 primary
 node (implicit) and 2 replicas (replicasPerNodeGroup)
        NumNodeGroups=NumNodeGroups,
        ReplicasPerNodeGroup=ReplicasPerNodeGroup,
        CacheParameterGroupName='default.valkey7.2.cluster.on'
    )

    return response


# Creates a cluster mode enabled
response = create_cluster_mode_enabled(
    CacheNodeType='cache.m6g.large',
    EngineVersion='6.0',
    ReplicationGroupDescription='Valkey cluster mode enabled with replicas',
    ReplicationGroupId='valkey20210',
#   Creates a cluster mode enabled cluster with 1 shard(NumNodeGroups), 1 primary
 (implicit) and 2 replicas (replicasPerNodeGroup)
    NumNodeGroups=2,
    ReplicasPerNodeGroup=1,
)

logging.info(response)
```

To run the program, enter the following command:

```
python ClusterModeEnabled.py
```

For more information, see Managing clusters in ElastiCache.

**Create a cluster mode enabled cluster with TLS and RBAC**

To ensure security, you can use Transport Layer Security (TLS) and Role-Based Access Control (RBAC) when creating a cluster mode enabled cluster. Unlike Valkey or Redis OSS AUTH, where all authenticated clients have full replication group access if their token is authenticated, RBAC enables you to control cluster access through user groups. These user groups are designed as a way to organize access to replication groups. For more information, see Role-Based Access Control (RBAC).

Copy the following program and paste it into a file named *ClusterModeEnabledWithRBAC.py*.

```
import boto3
```

```
import logging

logging.basicConfig(level=logging.INFO)
client = boto3.client('elasticache')

def
 create_cluster_mode_enabled(CacheNodeType='cache.t3.small',EngineVersion='6.0',NumNodeGroups=1
 ReplicationGroupDescription='Sample cache with cluster
 mode enabled',ReplicationGroupId=None,UserGroupIds=None,
 SecurityGroupIds=None,CacheSubnetGroupName=None,CacheParameterGroupName='default.valkey7.2.clu
    """Creates an ElastiCache Cluster with cluster mode enabled and RBAC

    Returns a dictionary with the API response

    :param CacheNodeType: Node type used on the cluster. If not specified,
 cache.t3.small will be used
    Refer to https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/
CacheNodes.SupportedTypes.html for supported node types
    :param EngineVersion: Engine version to be used. If not specified, latest will be
 used.
    :param NumNodeGroups: Number of shards in the cluster. Minimum 1 and maximun 90.
    If not specified, cluster will be created with 1 shard.
    :param ReplicasPerNodeGroup: Number of replicas per shard. If not specified 1
 replica per shard will be created.
    :param ReplicationGroupDescription: Description for the cluster.
    :param ReplicationGroupId: Name for the cluster.
    :param CacheParameterGroupName: Parameter group to be used. Must be compatible with
 the engine version and cluster mode enabled.
    :return: dictionary with the API results

    """
    if not ReplicationGroupId:
        return 'ReplicationGroupId parameter is required'
    elif not isinstance(UserGroupIds,(list)):
        return {'Error': 'UserGroupIds parameter is required and must be a list'}

    params={'AutomaticFailoverEnabled': True,
            'CacheNodeType': CacheNodeType,
            'Engine': 'valkey',
            'EngineVersion': EngineVersion,
            'ReplicationGroupDescription': ReplicationGroupDescription,
            'ReplicationGroupId': ReplicationGroupId,
            'SnapshotRetentionLimit': 30,
            'TransitEncryptionEnabled': True,
```

```
            'UserGroupIds':UserGroupIds,
            'NumNodeGroups': NumNodeGroups,
            'ReplicasPerNodeGroup': ReplicasPerNodeGroup,
            'CacheParameterGroupName': CacheParameterGroupName
        }

    # defaults will be used if CacheSubnetGroupName or SecurityGroups are not explicit.
    if isinstance(SecurityGroupIds,(list)):
        params.update({'SecurityGroupIds':SecurityGroupIds})
    if CacheSubnetGroupName:
        params.update({'CacheSubnetGroupName':CacheSubnetGroupName})

    response = client.create_replication_group(**params)
    return response

if __name__ == '__main__':
    # Creates a cluster mode enabled cluster
    response = create_cluster_mode_enabled(
        CacheNodeType='cache.m6g.large',
        EngineVersion='7.2',
        ReplicationGroupDescription='Valkey cluster mode enabled with replicas',
        ReplicationGroupId='valkey2021',
    #   Creates a cluster mode enabled cluster with 1 shard(NumNodeGroups), 1 primary
 (implicit) and 2 replicas (replicasPerNodeGroup)
        NumNodeGroups=2,
        ReplicasPerNodeGroup=1,
        UserGroupIds=[
            'mygroup'
        ],
        SecurityGroupIds=[
            'sg-7cc73803'
        ],
        CacheSubnetGroupName='default'

    )

    logging.info(response)
```

To run the program, enter the following command:

```
python ClusterModeEnabledWithRBAC.py
```

For more information, see Managing clusters in ElastiCache.

### Check if users/usergroup exists, otherwise create them

With RBAC, you create users and assign them specific permissions by using an access string. You assign the users to user groups aligned with a specific role (administrators, human resources) that are then deployed to one or more ElastiCache for Redis OSS replication groups. By doing this, you can establish security boundaries between clients using the same Valkey or Redis OSS replication group or groups and prevent clients from accessing each other's data. For more information, see Role-Based Access Control (RBAC).

Copy the following program and paste it into a file named *UserAndUserGroups.py*. Update the mechanism for supplying credentials. Credentials in this example are shown as replaceable and assigned an undeclared item. Avoid hard-coding credentials.

This example uses an access string with the permissions for the user. For more information on access strings see Specifying Permissions Using an Access String.

```
import boto3
import logging

logging.basicConfig(level=logging.INFO)
client = boto3.client('elasticache')

def check_user_exists(UserId):
    """Checks if UserId exists

    Returns True if UserId exists, otherwise False
    :param UserId: ElastiCache User ID
    :return: True|False
    """
    try:
        response = client.describe_users(
            UserId=UserId,
        )
        if response['Users'][0]['UserId'].lower() == UserId.lower():
            return True
    except Exception as e:
        if e.response['Error']['Code'] == 'UserNotFound':
            logging.info(e.response['Error'])
            return False
        else:
            raise
```

```
def check_group_exists(UserGroupId):
    """Checks if UserGroupID exists

    Returns True if Group ID exists, otherwise False
    :param UserGroupId: ElastiCache User ID
    :return: True|False
    """

    try:
        response = client.describe_user_groups(
            UserGroupId=UserGroupId
        )
        if response['UserGroups'][0]['UserGroupId'].lower() == UserGroupId.lower():
            return True
    except Exception as e:
        if e.response['Error']['Code'] == 'UserGroupNotFound':
            logging.info(e.response['Error'])
            return False
        else:
            raise

def create_user(UserId=None,UserName=None,Password=None,AccessString=None):
    """Creates a new user

    Returns the ARN for the newly created user or the error message
    :param UserId: ElastiCache user ID. User IDs must be unique
    :param UserName: ElastiCache user name. ElastiCache allows multiple users with the
 same name as long as the associated user ID is unique.
    :param Password: Password for user. Must have at least 16 chars.
    :param AccessString: Access string with the permissions for the user.
    :return: user ARN
    """
    try:
        response = client.create_user(
            UserId=UserId,
            UserName=UserName,
            Engine='Redis',
            Passwords=[Password],
            AccessString=AccessString,
            NoPasswordRequired=False
        )
        return response['ARN']
    except Exception as e:
        logging.info(e.response['Error'])
```

```
        return e.response['Error']

def create_group(UserGroupId=None, UserIds=None):
    """Creates a new group.
    A default user is required (mandatory) and should be specified in the UserIds list

    Return: Group ARN
    :param UserIds: List with user IDs to be associated with the new group. A default
 user is required
    :param UserGroupId: The ID (name) for the group
    :return: Group ARN
    """
    try:
        response = client.create_user_group(
            UserGroupId=UserGroupId,
            Engine='Redis',
            UserIds=UserIds
        )
        return response['ARN']
    except Exception as e:
        logging.info(e.response['Error'])


if __name__ == '__main__':

    groupName='mygroup2'
    userName = 'myuser2'
    userId=groupName+'-'+userName

    # Creates a new user if the user ID does not exist.
    for tmpUserId,tmpUserName in [ (userId,userName), (groupName+'-
default','default')]:
        if not check_user_exists(tmpUserId):
            response=create_user(UserId=tmpUserId,
 UserName=EXAMPLE,Password=EXAMPLE,AccessString='on ~* +@all')
            logging.info(response)
        # assigns the new user ID to the user group
    if not check_group_exists(groupName):
        UserIds = [ userId , groupName+'-default']
        response=create_group(UserGroupId=groupName,UserIds=UserIds)
        logging.info(response)
```

To run the program, enter the following command:

```
python UserAndUserGroups.py
```

## Tutorial: Connecting to ElastiCache

The following examples use the Valkey or Redis OSS client to connect to ElastiCache.

### Topics


- Connecting to a cluster mode disabled cluster
- Connecting to a cluster mode enabled cluster
- Connecting using IAM authentication


### Connecting to a cluster mode disabled cluster

Copy the following program and paste it into a file named *ConnectClusterModeDisabled.py*. Update the mechanism for supplying credentials. Credentials in this example are shown as replaceable and assigned an undeclared item. Avoid hard-coding credentials.

```
from redis import Redis
import logging

logging.basicConfig(level=logging.INFO)
redis = Redis(host='primary.xxx.yyyyyy.zzz1.cache.amazonaws.com', port=6379,
 decode_responses=True, ssl=True, username=example, password=EXAMPLE)

if redis.ping():
    logging.info("Connected to Redis")
```

To run the program, enter the following command:

```
python ConnectClusterModeDisabled.py
```

### Connecting to a cluster mode enabled cluster

The cluster client is included in redis-py; the standalone redis-py-cluster package is no longer required. Copy the following Python program and paste it into a file named *ConnectClusterModeEnabled.py*:

```
from redis.cluster import RedisCluster
```

```
import logging

logging.basicConfig(level=logging.INFO)
redis = RedisCluster(host="xxx.yyy.clustercfg.zzz1.cache.amazonaws.com", port=6379,
 decode_responses=True)

if redis.ping():
    logging.info("Connected to Redis")
```

To run the program, enter the following command:

```
python ConnectClusterModeEnabled.py
```

**Connecting using IAM authentication**

You can authenticate a Python connection using an AWS Identity and Access Management (IAM) identity instead of a user password. IAM authentication is available for caches running Valkey 7.2 and above or Redis OSS version 7.0 and above. This method requires Transport Layer Security (TLS) for in-transit encryption. Before you connect, create an IAM-enabled ElastiCache user and grant the `elasticache:Connect` action to the cache and user. For instructions on setting up IAM authentication, see Authenticating with IAM in ElastiCache.

The following Python program generates a short-lived IAM authentication token with an AWS Signature Version 4 pre-signed request and uses the token as the password. Copy the program and paste it into a file named *ConnectWithIAM.py*:

```
import boto3
import logging
from botocore.auth import SigV4QueryAuth
from botocore.awsrequest import AWSRequest
from urllib.parse import urlencode
from redis import Redis

logging.basicConfig(level=logging.INFO)

def generate_iam_auth_token(user_id, cache_name, region, is_serverless=True):
    """Generate a short-lived IAM authentication token. The token is valid for 15
 minutes."""
    credentials = boto3.Session().get_credentials().get_frozen_credentials()
    query_params = {"Action": "connect", "User": user_id}
    if is_serverless:
        query_params["ResourceType"] = "ServerlessCache"
```

```
    request = AWSRequest(method="GET", url=f"http://{cache_name}/?
{urlencode(query_params)}")
    SigV4QueryAuth(credentials, "elasticache", region, expires=900).add_auth(request)
    # The IAM authentication token is the signed URL with the scheme removed.
    return request.url[len("http://"):]

# For IAM-enabled users, the user name and user ID must be identical.
user_id = "iam-user-01"
# Cache names are converted to lowercase at creation. Supply the name in lowercase.
cache_name = "cache-01"
region = "us-east-1"

# IAM authentication requires TLS. Pass the IAM authentication token as the password.
token = generate_iam_auth_token(user_id, cache_name, region, is_serverless=True)
redis = Redis(host="cache-01.xxxxxx.serverless.use1.cache.amazonaws.com", port=6379,
 ssl=True, username=user_id, password=token, decode_responses=True)

if redis.ping():
    logging.info("Connected to ElastiCache with IAM authentication")
```

To run the program, enter the following command:

```
python ConnectWithIAM.py
```

> **Note**
>
> The IAM authentication token is valid for 15 minutes. For long-lived connections, regenerate the token before it expires. For a node-based (cluster) cache instead of a Serverless cache, call `generate_iam_auth_token` with `is_serverless=False`.

## Usage examples

The following examples use the boto3 SDK for ElastiCache to work with ElastiCache for Redis OSS.

### Topics

- Set and Get strings
- Set and Get a hash with multiple items
- Publish (write) and subscribe (read) from a Pub/Sub channel
- Write and read from a stream

### Set and Get strings

Copy the following program and paste it into a file named *SetAndGetStrings.py*.

```
import time
import logging
logging.basicConfig(level=logging.INFO,format='%(asctime)s: %(message)s')

keyName='mykey'
currTime=time.ctime(time.time())

# Set the key 'mykey' with the current date and time as value.
# The Key will expire and removed from cache in 60 seconds.
redis.set(keyName, currTime, ex=60)

# Sleep just for better illustration of TTL (expiration) value
time.sleep(5)

# Retrieve the key value and current TTL
keyValue=redis.get(keyName)
keyTTL=redis.ttl(keyName)

logging.info("Key {} was set at {} and has {} seconds until expired".format(keyName,
 keyValue, keyTTL))
```

To run the program, enter the following command:

```
python SetAndGetStrings.py
```

### Set and Get a hash with multiple items

Copy the following program and paste it into a file named *SetAndGetHash.py*.

```
import logging
import time

logging.basicConfig(level=logging.INFO,format='%(asctime)s: %(message)s')

keyName='mykey'
keyValues={'datetime': time.ctime(time.time()), 'epochtime': time.time()}

# Set the hash 'mykey' with the current date and time in human readable format
 (datetime field) and epoch number (epochtime field).
redis.hset(keyName, mapping=keyValues)
```

```
# Set the key to expire and removed from cache in 60 seconds.
redis.expire(keyName, 60)

# Sleep just for better illustration of TTL (expiration) value
time.sleep(5)

# Retrieves all the fields and current TTL
keyValues=redis.hgetall(keyName)
keyTTL=redis.ttl(keyName)

logging.info("Key {} was set at {} and has {} seconds until expired".format(keyName,
 keyValues, keyTTL))
```

To run the program, enter the following command:

```
python SetAndGetHash.py
```

**Publish (write) and subscribe (read) from a Pub/Sub channel**

Copy the following program and paste it into a file named *PubAndSub.py*.

```
import logging
import time

def handlerFunction(message):
    """Prints message got from PubSub channel to the log output

    Return None
    :param message: message to log
    """
    logging.info(message)

logging.basicConfig(level=logging.INFO)
redis = Redis(host="redis202104053.tihewd.ng.0001.use1.cache.amazonaws.com", port=6379,
 decode_responses=True)


# Creates the subscriber connection on "mychannel"
subscriber = redis.pubsub()
subscriber.subscribe(**{'mychannel': handlerFunction})

# Creates a new thread to watch for messages while the main process continues with its
 routines
```

```
thread = subscriber.run_in_thread(sleep_time=0.01)

# Creates publisher connection on "mychannel"
redis.publish('mychannel', 'My message')

# Publishes several messages. Subscriber thread will read and print on log.
while True:
    redis.publish('mychannel',time.ctime(time.time()))
    time.sleep(1)
```

To run the program, enter the following command:

```
python PubAndSub.py
```

**Write and read from a stream**

Copy the following program and paste it into a file named *ReadWriteStream.py*.

```
from redis import Redis
import redis.exceptions as exceptions
import logging
import time
import threading

logging.basicConfig(level=logging.INFO)

def writeMessage(streamName):
    """Starts a loop writting the current time and thread name to 'streamName'

    :param streamName: Stream (key) name to write messages.
    """
    fieldsDict={'writerId':threading.currentThread().getName(),'myvalue':None}
    while True:
        fieldsDict['myvalue'] = time.ctime(time.time())
        redis.xadd(streamName,fieldsDict)
        time.sleep(1)

def readMessage(groupName=None,streamName=None):
    """Starts a loop reading from 'streamName'
    Multiple threads will read from the same stream consumer group. Consumer group is
 used to coordinate data distribution.
    Once a thread acknowleges the message, it won't be provided again. If message
 wasn't acknowledged, it can be served to another thread.
```

```
    :param groupName: stream group were multiple threads will read.
    :param streamName: Stream (key) name where messages will be read.
    """

    readerID=threading.currentThread().getName()
    while True:
        try:
            # Check if the stream has any message
            if redis.xlen(streamName)>0:
                # Check if if the messages are new (not acknowledged) or not (already
 processed)
                streamData=redis.xreadgroup(groupName,readerID,
{streamName:'>'},count=1)
                if len(streamData) > 0:
                    msgId,message = streamData[0][1][0]
                    logging.info("{}: Got {} from ID
 {}".format(readerID,message,msgId))
                    #Do some processing here. If the message has been processed
 sucessfuly, acknowledge it and (optional) delete the message.
                    redis.xack(streamName,groupName,msgId)
                    logging.info("Stream message ID {} read and processed successfuly
 by {}".format(msgId,readerID))
                    redis.xdel(streamName,msgId)
            else:
                pass
        except:
            raise

        time.sleep(0.5)

# Creates the stream 'mystream' and consumer group 'myworkergroup' where multiple
 threads will write/read.
try:
    redis.xgroup_create('mystream','myworkergroup',mkstream=True)
except exceptions.ResponseError as e:
    logging.info("Consumer group already exists. Will continue despite the error:
 {}".format(e))
except:
    raise

# Starts 5 writer threads.
for writer_no in range(5):
    writerThread = threading.Thread(target=writeMessage, name='writer-'+str(writer_no),
 args=('mystream',),daemon=True)
```

```
    writerThread.start()

# Starts 10 reader threads
for reader_no in range(10):
    readerThread = threading.Thread(target=readMessage, name='reader-'+str(reader_no),
 args=('myworkergroup','mystream',),daemon=True)
    readerThread.daemon = True
    readerThread.start()

# Keep the code running for 30 seconds
time.sleep(30)
```

To run the program, enter the following command:

```
python ReadWriteStream.py
```

# Tutorial: Configuring Lambda to access ElastiCache in a VPC

In this tutorial you can learn how to create an ElastiCache serverless cache, create a Lambda function, then test the Lambda function, and optionally clean up after.

**Topics**


- Step 1: Creating an ElastiCache serverless cache.
- Step 2: Create a Lambda function for ElastiCache
- Step 3: Test the Lambda function with ElastiCache
- Step 4: Clean up (Optional)


## Step 1: Creating an ElastiCache serverless cache.

To create a serverless cache, follow these steps.

### Step 1.1: Create a serverless cache

In this step, you create a serverless cache in the default Amazon VPC in the us-east-1 region in your account using the AWS Command Line Interface (CLI). For information on creating serverless cache using the ElastiCache console or API, see Create a Valkey serverless cache.

```
aws elasticache create-serverless-cache \
  --serverless-cache-name cache-01 \
  --description "ElastiCache IAM auth application" \
```

```
  --engine valkey
```

Note that the value of the Status field is set to `CREATING`. ElastiCache creates your cache in about a minute.

## Step 1.2: Copy serverless cache endpoint

Verify that ElastiCache has finished creating the cache with the `describe-serverless-caches` command.

```
aws elasticache describe-serverless-caches \
  --serverless-cache-name cache-01
```

Copy the Endpoint Address shown in the output. You'll need this address when you create the deployment package for your Lambda function.

## Step 1.3: Create IAM Role

1. Create an IAM trust policy document, as shown below, for your role that allows the Lambda service to assume the new role. Save the policy to a file named *trust-policy.json*.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [{
           "Effect": "Allow",
           "Principal": {"Service": "lambda.amazonaws.com"},
           "Action": "sts:AssumeRole"
       }]
   }
   ```

2. Create an IAM policy document, as shown below. Save the policy to a file named *policy.json*.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "elasticache:Connect"
               ],
               "Resource": [
   ```

```
                "arn:aws:elasticache:us-
east-1:123456789012:serverlesscache:cache-01",
                "arn:aws:elasticache:us-east-1:123456789012:user:iam-user-01"
            ]
        }
    ]
}
```

3. Create an IAM role.

```
aws iam create-role \
--role-name "elasticache-iam-auth-app" \
--assume-role-policy-document file://trust-policy.json
```

4. Create the IAM policy.

```
aws iam create-policy \
  --policy-name "elasticache-allow-all" \
  --policy-document file://policy.json
```

5. Attach the IAM policy to the role.

```
aws iam attach-role-policy \
 --role-name "elasticache-iam-auth-app" \
 --policy-arn "arn:aws:iam::123456789012:policy/elasticache-allow-all"
```

## Step 1.4: Configure IAM authentication

**Important**

This step is required. Your Lambda function authenticates to ElastiCache using IAM. For this to work, you must create an IAM-enabled ElastiCache user, add it to a user group, and associate that user group with your cache. Without this configuration, connection attempts from Lambda will fail.

1. Create an IAM-enabled user. This is the user identity that your Lambda function will authenticate as.

```
aws elasticache create-user \
```

```
  --user-name iam-user-01 \
  --user-id iam-user-01 \
  --authentication-mode Type=iam \
  --engine valkey \
  --access-string "on ~* +@all"
```

2. Create a user group, add the user to it, and associate the group with your cache.

```
aws elasticache create-user-group \
  --user-group-id iam-user-group-01 \
  --engine valkey \
  --user-ids iam-user-01

aws elasticache modify-serverless-cache \
  --serverless-cache-name cache-01 \
  --user-group-id iam-user-group-01
```

# Step 2: Create a Lambda function for ElastiCache

To create a Lambda function to access the ElastiCache cache, take these steps.

## Step 2.1: Create a Lambda function

In this tutorial, we provide example code in Python for your Lambda function.

### Python

The following example Python code reads and writes an item to your ElastiCache cache. Copy the code and save it into a file named `app.py`. Be sure to replace the `elasticache_endpoint` value in the code with the endpoint address you copied in the previous step.

```
from typing import Tuple, Union
from urllib.parse import ParseResult, urlencode, urlunparse

import botocore.session
import valkey
from botocore.model import ServiceId
from botocore.signers import RequestSigner
from cachetools import TTLCache, cached
import uuid

class ElastiCacheIAMProvider(valkey.CredentialProvider):
```

```
    def __init__(self, user, cache_name, is_serverless=False, region="us-east-1"):
        self.user = user
        self.cache_name = cache_name
        self.is_serverless = is_serverless
        self.region = region

        session = botocore.session.get_session()
        self.request_signer = RequestSigner(
            ServiceId("elasticache"),
            self.region,
            "elasticache",
            "v4",
            session.get_credentials(),
            session.get_component("event_emitter"),
        )

    # Generated IAM tokens are valid for 15 minutes
    @cached(cache=TTLCache(maxsize=128, ttl=900))
    def get_credentials(self) -> Union[Tuple[str], Tuple[str, str]]:
        query_params = {"Action": "connect", "User": self.user}
        if self.is_serverless:
            query_params["ResourceType"] = "ServerlessCache"
        url = urlunparse(
            ParseResult(
                scheme="https",
                netloc=self.cache_name,
                path="/",
                query=urlencode(query_params),
                params="",
                fragment="",
            )
        )
        signed_url = self.request_signer.generate_presigned_url(
            {"method": "GET", "url": url, "body": {}, "headers": {}, "context": {}},
            operation_name="connect",
            expires_in=900,
            region_name=self.region,
        )
        # RequestSigner only seems to work if the URL has a protocol, but
        # Elasticache only accepts the URL without a protocol
        # So strip it off the signed URL before returning
        return (self.user, signed_url.removeprefix("https://"))

def lambda_handler(event, context):
```

```
    username = "iam-user-01" # replace with your user id
    cache_name = "cache-01" # replace with your cache name
    elasticache_endpoint = "cache-01-xxxxx.serverless.use1.cache.amazonaws.com" #
replace with your cache endpoint
    creds_provider = ElastiCacheIAMProvider(user=username, cache_name=cache_name,
is_serverless=True)
    valkey_client = valkey.Valkey(host=elasticache_endpoint, port=6379,
credential_provider=creds_provider, ssl=True, ssl_cert_reqs="none")

    key='uuid'
    # create a random UUID - this will be the sample element we add to the cache
    uuid_in = uuid.uuid4().hex
    valkey_client.set(key, uuid_in)
    result = valkey_client.get(key)
    decoded_result = result.decode("utf-8")
    # check the retrieved item matches the item added to the cache and print
    # the results
    if decoded_result == uuid_in:
        print(f"Success: Inserted {uuid_in}. Fetched {decoded_result} from Valkey.")
    else:
        raise Exception(f"Bad value retrieved. Expected {uuid_in}, got
{decoded_result}")

    return "Fetched value from Valkey"
```

This code uses the Python valkey-py library to put items into your cache and retrieve them. This code uses cachetools to cache generated IAM Auth tokens for 15 mins. To create a deployment package containing valkey-py and cachetools, carry out the following steps.

In your project directory containing the app.py source code file, create a folder package to install the valkey-py and cachetools libraries into.

```
mkdir package
```

Install valkey-py, cachetools using pip.

```
pip install --target ./package valkey
pip install --target ./package cachetools
```

Create a .zip file containing the valkey-py and cachetools libraries. In Linux and macOS, run the following command. In Windows, use your preferred zip utility to create a .zip file with the valkey-py and cachetools libraries at the root.

```
cd package
zip -r ../my_deployment_package.zip .
```

Add your function code to the .zip file. In Linux and macOS, run the following command. In Windows, use your preferred zip utility to add app.py to the root of your .zip file.

```
cd ..
zip my_deployment_package.zip app.py
```

### Step 2.2: Create the IAM role (execution role)

Attach the AWS managed policy named `AWSLambdaVPCAccessExecutionRole` to the role.

```
aws iam attach-role-policy \
 --role-name "elasticache-iam-auth-app" \
 --policy-arn "arn:aws:iam::aws:policy/service-role/AWSLambdaVPCAccessExecutionRole"
```

### Step 2.3: Upload the deployment package (create the Lambda function)

In this step, you create the Lambda function (AccessValkey) using the create-function AWS CLI command.

From the project directory that contains your deployment package .zip file, run the following Lambda CLI `create-function` command.

For the role option, use the ARN of the execution role you created in the previous step. For the vpc-config enter comma separated lists of your default VPC's subnets and your default VPC's security group ID. You can find these values in the Amazon VPC console. To find your default VPC's subnets, choose **Your VPCs**, then choose your AWS account's default VPC. To find the security group for this VPC, go to **Security** and choose **Security groups**. Ensure that you have the us-east-1 region selected.

```
aws lambda create-function \
--function-name AccessValkey  \
--region us-east-1 \
--zip-file fileb://my_deployment_package.zip \
--role arn:aws:iam::123456789012:role/elasticache-iam-auth-app \
--handler app.lambda_handler \
--runtime python3.12  \
--timeout 30 \
```

```
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=default-
security-group-id
```

## Step 3: Test the Lambda function with ElastiCache

In this step, you invoke the Lambda function manually using the invoke command. When the Lambda function executes, it generates a UUID and writes it to the ElastiCache cache that you specified in your Lambda code. The Lambda function then retrieves the item from the cache.

1. Invoke the Lambda function (AccessValkey) using the AWS Lambda invoke command.

   ```
   aws lambda invoke \
   --function-name AccessValkey  \
   --region us-east-1 \
   output.txt
   ```

2. Verify that the Lambda function executed successfully as follows:

   - Review the output.txt file.
   - Verify the results in CloudWatch Logs by opening the CloudWatch console and choosing the log group for your function (/aws/lambda/AccessValkey). The log stream should contain output similar to the following:

     ```
     Success: Inserted 826e70c5f4d2478c8c18027125a3e01e. Fetched
      826e70c5f4d2478c8c18027125a3e01e from Valkey.
     ```

   - Review the results in the AWS Lambda console.

## Step 4: Clean up (Optional)

To clean up, take these steps.

### Step 4.1: Delete Lambda function

```
aws lambda delete-function \
 --function-name AccessValkey
```

### Step 4.2: Delete Serverless cache

Delete the cache.

```
aws elasticache delete-serverless-cache \
 --serverless-cache-name cache-01
```

Remove users and user group.

```
aws elasticache delete-user \
 --user-id default-user-disabled

aws elasticache delete-user \
 --user-id iam-user-01

aws elasticache delete-user-group \
 --user-group-id iam-user-group-01
```

## Step 4.3: Remove IAM Role and policies

```
aws iam detach-role-policy \
 --role-name "elasticache-iam-auth-app" \
 --policy-arn "arn:aws:iam::123456789012:policy/elasticache-allow-all"

aws iam detach-role-policy \
--role-name "elasticache-iam-auth-app" \
--policy-arn "arn:aws:iam::aws:policy/service-role/AWSLambdaVPCAccessExecutionRole"

aws iam delete-role \
 --role-name "elasticache-iam-auth-app"

 aws iam delete-policy \
  --policy-arn "arn:aws:iam::123456789012:policy/elasticache-allow-all"
```

# Agent tools for ElastiCache

AI coding agents can use ElastiCache through purpose-built integrations from the AWS Agent Toolkit. These integrations provide service-specific guidance, reference material, and safety checks so an agent can evaluate, create, connect to, and operate ElastiCache workloads correctly.

This section covers the available agent tools and how to set them up.

**Topics**


- Work with Amazon ElastiCache using agent skills
- Work with Amazon ElastiCache using Valkey MCP

# Work with Amazon ElastiCache using agent skills

The Amazon ElastiCache skill in the AWS Agent Toolkit helps AI coding agents evaluate, create, connect, and operate ElastiCache workloads. Use the skill when you are using agent-assisted coding tools to work with ElastiCache for Valkey, Redis OSS, or Memcached workloads. The ElastiCache skill provides service-specific instructions, reference material, and safety checks to guide agents through the full ElastiCache lifecycle, from setup and data modeling to vector search and monitoring. The skill helps an AI coding agent achieve better task completion on ElastiCache tasks.

The skill covers use-cases like application caching, session storage, rate limiting, leaderboards, pub/sub, streams, durable caching, semantic caching for Agentic AI, AI agent memory, retrieval augmented generation, and vector search.

**Topics**


- Getting started
- How the ElastiCache skill works
- Guardrails and tracking usage


## Getting started

Install the ElastiCache skill using the following command:

```
npx skills add aws/agent-toolkit-for-aws/skills/specialized-skills/database-skills/
amazon-elasticache -g -a <agent-id> --full-depth
```

Replace `<agent-id>` in the install command with the identifier for your coding agent. The following table lists identifiers for supported coding agents. The installer adds the ElastiCache skill from the AWS Agent Toolkit repository to the global skills directory used by your coding agent, making it available for all your projects.

| Agent | <agent-id> | Global skills path | Project skills path |
|---|---|---|---|
| Claude Code | `claude-code` | `~/.claude/skills/` | `.claude/skills/` |

| Agent | <agent-id> | Global skills path | Project skills path |
|---|---|---|---|
| Cursor | cursor | ~/.cursor/skills/ | .agents/skills/ |
| GitHub Copilot | github-copilot | ~/.copilot/skills/ | .agents/skills/ |
| Codex | codex | ~/.codex/skills/ | .agents/skills/ |
| Gemini CLI | gemini-cli | ~/.gemini/skills/ | .agents/skills/ |
| Kiro CLI | kiro-cli | ~/.kiro/skills/ | .kiro/skills/ |
| Cline | cline | ~/.agents/skills/ | .agents/skills/ |
| Goose | goose | ~/.config/goose/skills/ | .goose/skills/ |
| Windsurf | windsurf | ~/.codeium/windsurf/skills/ | .windsurf/skills/ |

To install the skill locally for a single project instead, omit the `-g` flag. The skill installs to the project skills path listed in the preceding table.

> **Note**
>
> The skill runs locally with your agent. For tasks that create or modify AWS resources, your agent needs AWS credentials configured through the AWS Command Line Interface, environment variables, or your IDE's AWS plugin.

## Examples of usage

After installation, you can prompt your agent to build with the ElastiCache skill.

The following examples use "Use the Amazon ElastiCache skill to..." as a prefix:

- **Evaluate a workload**: `Use the Amazon ElastiCache skill to decide whether ElastiCache is right for my FastAPI app. The app reads heavily from Aurora PostgreSQL and has latency spikes during peak traffic.`
- **Create a secure cache**: `Use the Amazon ElastiCache skill to create a Valkey cache in us-east-1 for my ECS service. Use TLS and recommend the right authentication model.`
- **Connect an application**: `Use the Amazon ElastiCache skill to connect my Lambda function to an existing Valkey cache. Show the VPC, security group, and client code changes.`
- **Build a rate limiter**: `Use the Amazon ElastiCache skill to build an API rate limiter with Valkey. I need per-user limits and atomic updates.`
- **Add semantic caching for GenAI**: `Use the Amazon ElastiCache skill to add semantic caching for Amazon Bedrock responses. Include embedding storage, similarity lookup, thresholding, and fallback behavior.`
- **Troubleshoot performance**: `Use the Amazon ElastiCache skill to pinpoint the cause of high CPU utilization on my cache cluster.`

> **Note**
>
> Some agents auto-load the ElastiCache skill when your prompt describes a relevant workload such as caching, sessions, or rate limiting. If your agent does not load the skill automatically, include "use the ElastiCache skill" in your prompt.

## How the ElastiCache skill works

The ElastiCache skill routes your request to the appropriate workflow area based on your intent. It loads only the reference material needed for the current task. The following table describes the workflow areas included in the skill.

| Workflow area | Purpose |
| --- | --- |
| Solution fit | Gathers workload requirements and recommends whether ElastiCache is the right fit for your application. |
| Create and connect | Helps create caches, choose engines, configure networking, set up authentication, and connect applications. |
| Application patterns | Helps choose key schemas, TTLs, invalidation strategies, and Redis or Valkey data structures. |
| AI and vector workloads | Helps implement semantic caching, agent memory, RAG, recommendation, personali zation, and vector search patterns. |
| Operate and observe | Helps configure dashboards, alarms, logs, troubleshooting, cost reporting, and security audits. |
| Migration | Helps plan and run migrations from self-mana ged Redis, Redis OSS, or another ElastiCache deployment model. |

## Setup and networking

The skill helps agents identify the appropriate deployment configuration for a workload before generating setup guidance or commands. It guides agents to consider engine choice, serverless or node-based deployment, feature compatibility, and other requirements. The skill then guides agents through networking setup, including VPC placement, subnet and security group configuration, and application connectivity.

For local development, the skill guides agents to use supported access patterns such as a jump host or AWS Systems Manager Session Manager tunnel, instead of assuming direct access to an ElastiCache endpoint from a local machine.

The skill stores ElastiCache-specific project context in a state file at `.elasticache/requirements.json`. This file helps the agent avoid asking for the same information repeatedly. It can include values such as engine, AWS Region, and cache endpoints. If this file exists, the skill reads it before asking new questions.

## Cost estimation

The skill can help agents provide preliminary pricing guidance for ElastiCache workloads. The skill includes a pricing calculator to estimate monthly costs from workload inputs such as data size, request volume, command types, and payload size.

AWS credentials are not required to generate pricing estimates. The pricing calculator uses publicly available AWS pricing data, so you can estimate serverless and node-based costs before provisioning resources or connecting an AWS account. The pricing estimates are for planning purposes only and can vary from actual costs. Before deployment, confirm current pricing on the ElastiCache pricing page.

## Monitoring and troubleshooting

The monitoring workflow helps diagnose operational issues by starting with relevant metrics and then recommending the smallest effective change. The skill can use bundled scripts to collect metrics, generate dashboards, run security audits, and support cost analysis.

Use the ElastiCache skill to investigate performance, availability, and cost issues such as the following:

- High latency
- Low cache hit rates
- CPU or memory pressure
- Replication lag
- Connection spikes
- Hot keys and big keys
- Shard imbalance
- Slow commands
- Unexpected cost increases
- CloudWatch alarm configuration

- Log delivery

### GenAI and vector workload guidance

The skill helps agents provide guidance for AI application patterns such as semantic caching, AI agent memory, retrieval-augmented generation (RAG), recommendation systems, and vector search. The skill helps agents identify the user's intent, select the appropriate workflow, and generate implementations that are aligned with ElastiCache capabilities.

For these workloads, the skill provides detailed guidance for embedding storage, key design, TTL strategy, similarity threshold tuning, framework integrations, and operational considerations.

## Guardrails and tracking usage

The ElastiCache skill includes safety guidance for high-impact operations so agents do not treat destructive or disruptive changes as routine setup steps. The skill identifies high-impact operations that must not be run automatically, including cache deletion, flushing data, snapshot deletion, production credential or AUTH token changes, security group rule removal, user or user group deletion, migration cutover execution, and manual failover on production caches. For such operations, the skill instructs agents to explain the expected impact and require direct user confirmation before proceeding. For changes that can affect availability, such as engine upgrades, scaling changes, security group updates, failover tests, or migration cutovers, the skill instructs agents to explain the expected impact, recommend staging validation or a maintenance window where appropriate, and prefer safer alternatives when available.

Before generating commands that create, modify, or delete resources, the skill instructs agents to validate user-provided values. If required information is missing or ambiguous, the skill instructs agents to stop and ask for the missing values instead of generating commands with placeholders, inferred settings, or incomplete resource identifiers.

### Tracking agent activity

You can identify actions performed by an agent using the ElastiCache skill through the AWS CloudTrail `userAgent` field and ElastiCache resource tags.

For AWS API calls generated through the AWS Command Line Interface, AWS SDKs, or bundled scripts, the skill instructs agents to set `AWS_SDK_UA_APP_ID` to `AWSSkill-ElastiCache`. This identifier appears in the `User-Agent` HTTP header and in the CloudTrail `userAgent` field for each API call.

For resources created by an agent using the ElastiCache skill, review the resource tags in the ElastiCache console or by using `list-tags-for-resource`. The skill instructs agents to apply the following standard tags when creating resources:

- `managed_by=aws-skills`
- `skill=elasticache`
- `skill_version=<skill-version>`
- `created_by=elasticache-skill`
- `generation_model=<model-id>`

# Work with Amazon ElastiCache using Valkey MCP

Model Context Protocol (MCP) servers give AI agents direct access to tools and data through structured, standardized interfaces. The Valkey MCP server provides agents with direct access to Valkey operations, allowing them to inspect data, execute commands, and work with common Valkey data structures from natural-language requests.

**Topics**


- Getting started
- How Valkey MCP Server works
- Guardrails and tracking usage


## Getting started

### Installation

**Prerequisites:** Python 3.10 or later, uv, Access to a Valkey datastore

**Configure your MCP client:** Add the following to your MCP client configuration:

```
{
  "mcpServers": {
    "valkey": {
      "command": "uvx",
      "args": ["awslabs.valkey-mcp-server@latest"],
      "env": {
        "VALKEY_HOST": "<your-endpoint>",
        "VALKEY_USE_SSL": "true"
      }
    }
  }
}
```

Replace `<your-endpoint>` with the endpoint of your Valkey datastore. Your coding agent must have network access to the endpoint.

**Connecting to Amazon ElastiCache:** If your Valkey serverless cache has a public endpoint, use the public endpoint address as `VALKEY_HOST`. No tunnel is required. The connection requires TLS

1.3 and an IAM authentication token. For details on generating the token, see the section called "Connect to your cache".

For all other ElastiCache for Valkey caches, you cannot reach the cache directly from your local machine. For SSM tunnel setup, IAM auth token generation, and TLS handling in tunnel mode, see the ElastiCache connection guide in the AWS Agent Toolkit, or install the Amazon ElastiCache Skill to let your agent handle the connection setup end-to-end. After the tunnel is established, use `VALKEY_HOST=127.0.0.1` in the config above.

Restart your MCP client after saving the configuration.

You can prompt your agent to verify the connection with:

*Use the Valkey MCP server to verify the connection to my datastore and confirm that Valkey operations are available.*

## Configuration

Configure the Valkey MCP server through environment variables in your MCP client configuration. You can also use the `--readonly` flag to restrict agent access to read-only operations.

| Variable | Purpose |
|---|---|
| `VALKEY_HOST` | Specifies the endpoint of your Valkey datastore. Use `127.0.0.1` when connecting to a private ElastiCache cluster through a tunnel. |
| `VALKEY_USE_SSL` | Set to `true` to enable TLS. This is required for ElastiCache clusters with encryption in transit enabled. |
| `--readonly` | Restricts the server to the read tier and disables all write and administrative tools, making it suitable for production exploration. |
| **Optional Search Parameters** | |
| `EMBEDDINGS_PROVIDER` | Selects the embedding provider used by `add_documents` and `search`. Supported |

| Variable | Purpose |
| --- | --- |
| | options are `bedrock`, `openai`, `ollama`, and `hash`. Use `hash` for local testing without credentials. |
| `BEDROCK_MODEL_ID` | Overrides the default Amazon Bedrock embedding model. |
| `OPENAI_API_KEY` | Provides the API key required when `EMBEDDING_PROVIDER` is set to `openai`. |

## How Valkey MCP Server works

The Valkey MCP server gives your agent a set of structured tools for working with the configured Valkey datastore. When you make a request, the agent selects the appropriate tool, provides the required parameters, and receives the result directly from Valkey. The server handles command construction and validation, so the agent does not need to generate client code or manually format Valkey commands.

The Valkey MCP server provides the following capability areas:

| Capability area | Purpose |
| --- | --- |
| **Valkey operations** | Read and modify strings, hashes, lists, sets, sorted sets, streams, and other Valkey data structures. |
| **JSON operations** | Store, retrieve, and update JSON documents and arrays using structured paths. |
| **Search and AI workloads** | Create search indexes, add documents, run text, semantic, vector, and hybrid searches, and aggregate results. |
| **Operational safeguards** | Separates read, write, and administrative operations. Destructive administrative |

| Capability area | Purpose |
| --- | --- |
| | commands are disabled by default, and the server can be run in read-only mode. |

The server uses structured JSON inputs and translates them into the appropriate Valkey operations. This reduces command-formatting errors and gives the agent a consistent interface across common data, search, and operational tasks.

## Full Valkey access through a compact tool surface

The Valkey MCP server gives agents access to the full Valkey command set through three structured tools:

1. `valkey_read` handles read-only operations such as `GET`, `HGETALL`, `SCAN`, and `INFO`.
2. `valkey_write` supports mutations such as `SET`, `HSET`, `DEL`, and `LPUSH`.
3. `valkey_admin` covers configuration and high-impact operations.

This keeps the tool surface simple for agents while supporting Valkey capabilities like strings, hashes, lists, sorted sets, streams, pub/sub, bitmaps, HyperLogLog, and others.

## Simplified AI, search, and analytics workflows

The Valkey MCP server provides four purpose-built tools for vector, semantic, text, and hybrid search, along with aggregations. Agents can query your data without constructing low-level commands, giving developers a direct path from a natural-language request to a working Valkey search or AI workflow.

1. `manage_index` lets agents create, drop, inspect, or list search indexes.
2. `add_documents` ingests documents into a search index and automatically creates the index if it does not exist. It also supports optional embedding generation for semantic search, using open-source embedding packages by default and configurable providers such as Amazon Bedrock, OpenAI, or Ollama.
3. `search` provides a unified interface for semantic, text, hybrid, and find-similar searches. It automatically detects the search mode from the parameters provided.
4. `aggregate` provides a structured pipeline builder for `FT.AGGREGATE`, with support for `GROUPBY`, `SORTBY`, `APPLY`, `FILTER`, and `LIMIT` stages.

When debugging, the `search` and `aggregate` tools let agents inspect the actual contents of your datastore. Agents can review indexed data, sample matching documents, and validate search and aggregation results directly in the conversation.

## Guardrails and tracking usage

The Valkey MCP server uses a risk-tiered execution model that separates read, write, and administrative operations. The read tier is always available and provides the lowest-risk level of access. The write tier allows agents to modify data while continuing to block destructive operations such as `FLUSHALL`. The admin tier covers high-impact operations and is disabled by default. Enabling it requires both a server-level setting and explicit confirmation for each call.

For production exploration, you can also run the server with `--readonly` to restrict agents to read-only operations.

### Track agent activity

The Valkey MCP server writes a structured log of every tool call to `valkey-mcp-server.log` in your MCP client's working directory, showing the operation, parameters, and result. Open this file to see what your agent is doing against your datastore in real time.

# Migrating to ElastiCache

With online migration, you point ElastiCache at your running self-managed source, and ElastiCache replicates your data into a target cache while your source keeps serving.

Online migration does not move data between two ElastiCache caches.

Choose your path by target. If your target is an ElastiCache Serverless cache, use the serverless path. If your target is a node-based ElastiCache cluster that you size and manage yourself, use the node-based path.

Both paths require a self-managed source that runs on Amazon EC2.

**Topics**


- Online migration to ElastiCache Serverless
- Online migration to ElastiCache node-based clusters


## Online migration to ElastiCache Serverless

Amazon ElastiCache Serverless for Valkey supports migration from self-managed Valkey and Redis OSS on Amazon EC2 with zero downtime.

> **Zero downtime**
>
> Zero downtime means zero Errors returned to the client during migration, measured the same way the ElastiCache SLA measures availability. Your application continues reading and writing through cutover.

These topics cover migration to an ElastiCache Serverless cache. To migrate to a node-based cluster instead, see the section called “Online migration to ElastiCache node-based clusters”.

For more information about ElastiCache Serverless, see Serverless caching.

Online migration moves your self-managed Valkey or Redis OSS data into ElastiCache Serverless for Valkey while your application keeps serving traffic. You do not need a third-party replication tool or a maintenance window to switch over.

Migration usage is free for the first 180 days per AWS account, starting when the account first prepares a migration. Your target ElastiCache Serverless cache bills at the standard rates during the migration. For more information, see Amazon ElastiCache pricing on the AWS website.

During a migration, you point your application at the migration endpoint. The migration endpoint is your serverless cache's standard endpoint, so your application keeps using it after cutover. ElastiCache routes your application's requests to your source cache while it replicates your data to the target cache. When you complete the cutover, ElastiCache switches over at the next fully synced window, and your application is then served by ElastiCache Serverless.

You can stop the data sync while it is in progress. Your application continues to be served by your source cache.

The following topics describe the requirements, source authentication, the console and AWS CLI procedure, and cutover modes for online migration.

**Topics**


- How a migration works
- Migration states
- Monitoring a migration
- How you learn that a migration finished
- Supported sources and targets
- Errors returned by the migration operations
- Prerequisites for online migration to ElastiCache Serverless
- Source authentication for online migration to ElastiCache Serverless
- Migrate your data to ElastiCache Serverless
- Cutover modes for online migration to ElastiCache Serverless


## How a migration works

A migration has three steps, each with its own API operation: prepare the migration, start the data sync, and complete the cutover. ElastiCache publishes an event at every transition.

1. **Prepare the migration.** ElastiCache validates your source cache and your target cache against the migration requirements. It provisions the infrastructure that reads from your source and

returns the migration endpoint. You then point your application at the migration endpoint and confirm that it is serving traffic through the endpoint. The console reminds you to do this before you start the data sync, because ElastiCache cannot detect whether your application has moved. For more information, see Prerequisites for online migration to ElastiCache Serverless and Redirect your application.

2. **Start the data sync.** ElastiCache begins replicating data from your source cache to the target cache. It keeps replicating as your application writes. For more information, see Start the data sync.
3. **Complete the cutover.** ElastiCache switches your migration endpoint over to serve from the target cache at the next fully synced window. You choose whether to trigger this yourself or have ElastiCache do it for you. For more information, see Cutover modes for online migration to ElastiCache Serverless.

For the console and AWS CLI steps, see Migrate your data to ElastiCache Serverless.

## Migration states

You can check the current state at any time with `describe-serverless-caches`. This command returns a `Migration` object on the cache, which is the record of your migration that `describe-serverless-caches` returns. The migration can be in one of the following states:

| State | What the console shows | What it means | What you can do |
|---|---|---|---|
| `preparing` | Preparing migration | ElastiCache is validating your source cache and provisioning migration infrastructure. | Wait for Ready to sync. |
| `ready-to-sync` | Ready to sync | You can route your traffic through the migration endpoint and start the data sync. | Choose **Start migration** to start the data sync. **Force complete** is available. **Stop** is disabled. |

| State | What the console shows | What it means | What you can do |
|---|---|---|---|
| `initiating-sync` | Initializing sync | ElastiCache is starting replication across all shards. | Wait for Data sync in progress. |
| `syncing` | Data sync in progress | Replication is in progress. | Choose **Complete migration**. **Complete migration** is available only when you chose **Manual cutover**. With **Auto-cutover**, it is disabled, because ElastiCache cuts over at the next fully synced window. **Stop** and **Force complete** are available. |
| `cutting-over` | Cutover in progress | ElastiCache is finalizing the cutover to the target cache. | Wait for the cutover to finish. **Stop** and **Force complete** are disabled. |
| `stopping` | Stopping | ElastiCache is stopping the data sync at your request. Traffic continues through the migration endpoint. When the data sync stops, the migration returns to Ready to sync. | Wait for Ready to sync. |

The primary action button on the migration details page changes its label with the state. It reads Start migration until the data sync begins, and Complete migration afterward, in the same position.

How long the prepare step takes depends on the size of your source, because ElastiCache provisions migration infrastructure to match it. While the migration state reads `preparing`, the step is still running, so wait for `ready-to-sync`. After you start the data sync, the migration stays in `initiating-sync` until replication begins.

You can check the status with the AWS CLI. Run the following command:

```
aws elasticache describe-serverless-caches \
    --serverless-cache-name my-cache \
    --query "ServerlessCaches[0].[Status,Migration.Status]" \
    --output text
```

The command returns the cache status and the migration status:

| Cache status | Migration status | What it means |
|---|---|---|
| `migrating` | `preparing` | The prepare step is running. |
| `migrating` | `ready-to-sync` | The prepare step succeeded. |
| `migrating` | `syncing` | The data sync is running. |
| `available` | None | The migration ended. It completed, it was force completed, or the prepare step failed, so read the events. |

The response also returns the migration endpoint address and port, along with a `CutoverMode` field. `CutoverMode` reports whether you opted in to automatic cutover when you started the data sync. It does not indicate that a cutover is in progress.

**Console labeling**

On the migration details page, the **Migration endpoint** field shows this address, and **Copy migration endpoint** copies it. The console hides the field while the migration is in Preparing migration, and shows it in every later state.

The Migrations page shows the current state only. It shows no warning, no error badge, and no failure state. When a step fails, the state moves backward to the previous state, and a failed prepare step removes the migration from the page. These two changes are the only failure signals on this page. A stop that you request also moves the state backward.

To find the reason, open the cache details page for your target cache and read the events on its Events tab.

ElastiCache spends most of the `cutting-over` state tearing down the migration infrastructure. With Manual cutover, the state starts when you complete the migration, and ElastiCache first waits for a fully synced window to switch your traffic to your target. With Auto-cutover, the state stays `syncing` until ElastiCache finds that window. After the switch, your traffic is on your target for the rest of the state.

**Note**

Before you start the migration, the **Cutover method** field on the migration details page shows the API default, which can render as **Manual (default)**. It does not reflect a choice you have made. You choose the mode in the **Start migration** dialog box, where the control is labeled **Cutover mode**, with the values **Manual cutover** and **Auto-cutover**.

**Note**

On the Valkey caches page, the **Status** column shows **Migrating** for a cache that is the target of a migration. This is a cache status, not a migration state.

## Monitoring a migration

ElastiCache publishes an event at each phase transition. The transitions are the following:

- Prepare succeeded or failed
- Start succeeded or failed
- Cutover started
- Complete succeeded or failed
- Stop succeeded or failed

Failure events carry the reason.

For a migration that succeeds, the events read as follows, where `my-cache` is the name of your target cache:

| Transition | Event message |
| --- | --- |
| Prepare succeeded | Successfully prepared serverless cache *my-cache* for migration. |
| Start succeeded | Successfully started data sync for serverless cache *my-cache*. |
| Cutover started | Cutover initiated for serverless cache *my-cache*. Waiting for a suitable window to finalize the migration. |
| Complete succeeded | Successfully completed the migration for serverless cache *my-cache*. |
| Stop succeeded | Successfully stopped data sync for serverless cache *my-cache*. |
| Force complete succeeded | Successfully completed the migration for serverless cache *my-cache* with Force. |

When a step fails, the event names the step and gives the reason:

- Failed to prepare migration for serverless cache *my-cache*. *reason*
- Failed to start data sync for serverless cache *my-cache*. *reason*

- Failed to complete the migration for serverless cache *my-cache*. *reason*
- Failed to stop the migration for serverless cache *my-cache*. *reason*
- Migration error for serverless cache *my-cache*. *reason*

For example, a prepare step on a target that contains data reports Failed to prepare migration for serverless cache *my-cache*. The serverless cache must have no keys before migration can begin.

To list the events with the AWS CLI, run the following command:

```
aws elasticache describe-events \
    --source-type serverless-cache \
    --source-identifier my-cache \
    --duration 120
```

Where:

- `my-cache` is the name of your target cache.
- `120` is the number of minutes of events to return.

To follow a migration's progress, check the migration state that `describe-serverless-caches` returns. Use the events to learn the reason when a step fails. An event can appear up to about a minute after the state changes.

You read a migration's events on the target cache's Events tab, or on the Events page in the console navigation. The Events tab filter offers a fixed set of predefined groups, and no migration-specific filter is available. The Migrations page shows migrations that are in progress. A completed migration no longer appears there. To review a finished migration, read the target cache's events.

The console reads the migration state from `DescribeServerlessCaches`, which returns the `Migration` object on your target cache. It uses `DescribeEvents` to refresh the Monitor tab. The console can lag the underlying state.

There are no metrics specific to migration. You watch the target cache with the standard ElastiCache Serverless metrics. The migration details page in the console has a Monitor tab. The Monitor tab shows nine of the target cache's existing metrics:

- `BytesUsedForCache`
- `CurrItems`

- `NewConnections`
- `NetworkBytesIn`
- `NetworkBytesOut`
- `GetTypeCmds`
- `SetTypeCmds`
- `Evictions`
- `CurrConnections`

To confirm that replication is running, run `INFO replication` on each primary of your source. The migration appears there as a connected replica with `state=online` and a `lag` value. For more information, see Metrics and events for Valkey and Redis OSS serverless caches.

The tab opens with an information banner, "Track your migration progress". It reads "These metrics show activity on your target cache. Use them to confirm your data is arriving during the migration and to verify traffic after cutover."

The Monitor tab shows no progress percentage and no progress bar. Migration progress can move backward.

Before the data sync starts, the tab shows an empty state.

## How you learn that a migration finished

You do not have to stay on the migration details page. The same status banner appears on the target cache's details page. With automatic cutover, the console takes you to the cache details page when the migration completes.

On success, the cache details page shows a banner that names the completion time and points you to that cache's Events tab. On failure, the same surface shows a failure banner that carries the event detail.

A success banner and a failure banner can both be present at once.

A banner stays until you dismiss it, and it does not come back once dismissed. The console shows outcome banners for migrations from the past 7 days.

A completed migration no longer appears on the Migrations page.

## Supported sources and targets

Online migration supports the following source and target combinations:

| Component | Supported |
| --- | --- |
| Source | Self-managed Valkey 7.2 through 8.1, or Redis OSS 4.0 through 7.2, running on Amazon EC2 in cluster mode. |
| Target | An ElastiCache Serverless for Valkey cache running Valkey 7.2, 8.1, or 9.0, that you create before you start. |

Your source version must be no higher than your target version. ElastiCache managed caches and Amazon MemoryDB caches are not supported as sources. Valkey 9.0 is supported as a target but not as a source.

Your source must run in cluster mode. To migrate a source that runs with cluster mode disabled, use online migration to a node-based cluster. For more information, see the section called “Online migration to ElastiCache node-based clusters”.

## Errors returned by the migration operations

`StartMigration` and `CompleteMigration` are rejected immediately when the migration is not healthy. The error tells you to check the cache's events for the reason.

The migration operations raise the following exceptions:

| Exception | When |
| --- | --- |
| `MigrationHealthFault` | The migration is not healthy. Set `Force` to true on `CompleteMigration` to bypass the check. |
| `ServerlessCacheAlreadyUnderMigrationFault` | The cache is already the target of a migration. |

| Exception | When |
| --- | --- |
| `ServerlessCacheNotUnderMigrationFault` | The cache has no migration to act on. |
| `InvalidServerlessCacheStateFault` | The cache or the migration is not in a state that allows the operation. |
| `InvalidParameterValue` | The request is not valid for the target cache or the migration, for example when the target has a public endpoint or uses the IPv6 or dual-stack network type, or when you call `CompleteMigration` without `Force` while the migration is `ready-to-sync` or an automatic cutover is pending. |

## Prerequisites for online migration to ElastiCache Serverless

Before you start an online migration, confirm that your source cache and your target cache meet the following requirements. ElastiCache checks these requirements when you prepare the migration and checks some of them again when you start the data sync. If a requirement is not met, the step fails. ElastiCache reports a failure event on the cache that names the requirement. If the prepare step fails, ElastiCache rolls back the migration, removes the `Migration` object and returns your cache to `available`. Correct the requirement, then try again.

### Source cache requirements

Your source cache must meet the following requirements. ElastiCache checks each one at the times shown.

| Requirement | What your source must meet | When ElastiCache checks it |
| --- | --- | --- |
| Engine and version | Valkey 7.2 through 8.1, or Redis OSS 4.0 through 7.2. Later versions are not supported as a source. | Prepare and start |

| Requirement | What your source must meet | When ElastiCache checks it |
|---|---|---|
| Version not higher than the target | No higher than the version of your target ElastiCache Serverless cache. Migrating from a higher source version to a lower target version is rejected. | Prepare and start |
| Cluster mode | Runs in cluster mode. Prepare rejects a source that has cluster mode disabled. If your source stops running in cluster mode after prepare, the migration fails. | Prepare, and during the migration |
| Deployment | A self-managed cache running on Amazon EC2. ElastiCache managed caches and Amazon MemoryDB caches are not supported as sources. | Prepare |
| Memory per node | Up to 32 GB of data, measured as `used_memory` | Prepare |
| Shards | Up to 166 shards | Prepare |
| Slot coverage | Hash slots cover the full range with no gaps and no overlaps. | Prepare |

| Requirement | What your source must meet | When ElastiCache checks it |
| --- | --- | --- |
| No resharding | Not undergoing a slot migration when you prepare the migration, and not undergoing one while the migration runs. A slot migration that begins mid-flight fails the migration. | Prepare, and during the migration |
| Node health | Nodes must be healthy. ElastiCache checks node state when you prepare and continues to monitor node state. | Prepare, and during the migration |
| Protected mode | Protected mode turned off (`protected-mode no` ) on a source that has no password. With protected mode on, a node without a password accepts only local connectio ns, so ElastiCache cannot connect to it. A source that requires a password can keep protected mode on. | Prepare |
| Replication commands | Your source must accept a replica connection from the migration path. It must support `PSYNC` and `REPLCONF`. If you renamed or disabled these commands on your source, restore them before you migrate. | Prepare and start |

| Requirement | What your source must meet | When ElastiCache checks it |
| --- | --- | --- |
| Key name size | No key name in your source can exceed 4 KiB, which is the ElastiCache Serverless limit. An oversized key name fails the migration during the initial data load, and you cannot start the data sync again. ElastiCache writes the key to your target before it reports the failure. To recover, see If a key name exceeds 4 KiB. | During the initial data load |
| CPU headroom on source primaries | Your source primaries serve your application's traffic for the whole migration, up until cutover. You need CPU headroom on your source primaries to serve your full read and write workload while the migration runs. Review your workload and scale out before you prepare the migration. A source primary that becomes overloaded can compromise the health of the migration. For more information about when traffic moves off your source, see Cutover modes for online migration to ElastiCache Serverless. | Not checked. Your responsibility. |

| Requirement | What your source must meet | When ElastiCache checks it |
| --- | --- | --- |
| Memory for the initial full sync | ElastiCache replicates your data with the replication protocol, which starts with a full sync. A full sync forks the process on your source primary. That fork needs memory available on the primary beyond what your dataset already uses. How much it needs rises with your write rate during the sync, because your source primary must hold writes that arrive during the fork. | Not checked. Your responsibility. |
| Stable environment | Your environment must stay stable for the whole migration. Three things must stay valid until the migration completes: your network path, your source TLS certificate, and your AWS Secrets Manager credentials. If any of them changes mid-migration, ElastiCache reports a migration error event, and the migration cannot be started again. For more information, see Cutover modes for online migration to ElastiCache Serverless. | During the migration |

| Requirement | What your source must meet | When ElastiCache checks it |
| --- | --- | --- |
| Authentication and encryption | If your source requires authentication, store the credentials in Secrets Manager and grant ElastiCache permission to read the secret. The user name and password in the secret must meet the ElastiCache requirements, whether you created the secret or ElastiCache created it for you. If your source requires TLS, provide the source TLS configuration. For more information, see Source authentication for online migration to ElastiCache Serverless. | Prepare and start |

Compatibility holds for Redis versions whose data types Valkey supports. Valkey forked from Redis 7.2.4, where the supported range ends.

A second, separate check applies while the migration is running. If your source dataset grows beyond what your target cache can hold, the migration fails with a memory error. Leave headroom if your dataset is growing.

## Target cache requirements

Create an ElastiCache Serverless for Valkey cache before you start. Your target must meet all of the following requirements.

| Requirement | What your target cache must meet |
| --- | --- |
| Available and idle | In the `available` state with no other operation in progress. |

| Requirement | What your target cache must meet |
| --- | --- |
| Empty | Contains no keys. Flush the target yourself before you prepare the migration. If the target contains data, preparation fails. |
| No active pub/sub connections | No subscribed clients when you prepare the migration. |
| No public endpoint | No public endpoint. Leave public connectivity turned off when you create the target cache. For a cache that already exists, confirm that it has no public endpoint before you prepare the migration. |
| Network type | IPv4. ElastiCache rejects a target cache that uses the IPv6 or dual-stack network type. |
| Version | Runs Valkey 7.2, 8.1, or 9.0, at a version at least as high as your source version. |
| Authentication | Accepts the credentials that your application uses on your source. If your source requires authentication, set up the same users on your target before you prepare the migration. For more information about setting up target users, see Setting up your application's users on your target. |

Your target inherits the standard ElastiCache Serverless limits, and your data must fit inside them:

| Limit | Value |
| --- | --- |
| Size per cache | 5,000 GiB |
| Size per slot | 32 GiB |
| ElastiCache processing units | 15,000,000 ECPUs per second |

| Limit | Value |
| --- | --- |
| Key name length | 4 KiB |

## Network requirements

ElastiCache connects to your source from the VPC of your target cache. Confirm the following:

1. Confirm that your source is reachable on a private endpoint from that VPC. Also confirm that your source security group allows inbound traffic from ElastiCache.
2. Your source nodes must listen on addresses that this VPC can reach. The default Valkey and Redis OSS configuration file binds to `127.0.0.1`, which accepts only local connections, so bind each node to its private IP address. Each node must also report a reachable address in its cluster topology, because ElastiCache discovers the rest of your nodes from the first one.
3. For the security group rule, allow inbound TCP traffic on your source port from the security group of your target cache, or from the IPv4 CIDR ranges of its subnets.
4. For the source address, enter the private IP address of a source node, or a host name that resolves through public DNS. A host name that resolves only in your VPC's private DNS fails the prepare step.
5. Create your target cache with at least three subnets in different Availability Zones, each with free IP addresses. You cannot change the subnets of a serverless cache after you create it. If the prepare step or the data sync fails on subnets, the error message names the Availability Zones that need a subnet. Because you cannot change the subnets of a serverless cache, create a new target cache with a subnet in each Availability Zone that the message names, and prepare the migration again.
6. Preparing a migration creates network interfaces in the subnets of your target cache. They use the security groups of your target cache, and ElastiCache deletes them when the migration completes. The prepare step fails if your target's subnets have no free IP addresses or your VPC has no network interface quota headroom.
7. Keep the Secrets Manager secret that you provided until the migration completes. If a node is replaced during the migration, ElastiCache runs network and authentication setup again, and a missing secret fails that step.

## Command behavior during a migration

While your application is served through the migration endpoint, ElastiCache Serverless command support applies, not your source's command support.

Commands that ElastiCache Serverless does not support are rejected at the migration endpoint, the same as they would be on any serverless cache.

Some commands behave differently on ElastiCache Serverless than on a self-managed cache. These include FLUSH, the CLUSTER commands, and the publish and subscribe commands. ElastiCache Serverless requires every Lua script call to declare at least one key. A call such as EVALSHA with zero keys returns `ERR Lua scripts without any input keys are not supported`. Review your application's use of these commands before you start. For more information, see Supported and restricted Valkey, Memcached, and Redis OSS commands.

**Writes during the migration**

Do not send writes directly to your source during the migration. Commands that ElastiCache Serverless does not support are not replicated. A client can bypass the migration endpoint and write an unsupported command straight to your source. That makes your target diverge. ElastiCache reports a migration error event when it detects this. After ElastiCache reports that error, you cannot start the migration again. To end it, you complete the migration with force, which accepts the inconsistency.

The following commands need your attention during a migration:

| What | What to do |
| --- | --- |
| Scripts | Scripts you loaded on your source before the migration are not replicated to your target. Re-issue `SCRIPT LOAD` for them through the migration endpoint any time after the migration reaches `ready-to-sync` and before cutover. A script that is missing on the target returns `NOSCRIPT` after cutover. |

| What | What to do |
| --- | --- |
| Pub/sub subscriptions | Pub/sub subscriptions do not survive cutover. Subscribed clients are disconnected at cutover and have to reconnect and re-subscribe. |

## Client configuration

Your application connects to the migration endpoint with TLS, because all ElastiCache Serverless caches use encryption in transit. If your application reaches your source without TLS, turn on TLS in your client before you point it at the migration endpoint. The security group of your target cache must allow inbound TCP traffic from your application on ports 6379 and 6380, because many clients connect to both.

Instead of caching node IP addresses in your application, use the migration endpoint.

Stay within Valkey's default limit of 10,000 client connections during a migration. If your workload needs more connections than that, open them gradually rather than all at once.

Because commands are serialized during a migration, a client that relies on pipelining needs more concurrent connections to sustain the same request rate. If your client uses a connection pool, review its size before you migrate, and consider raising it for the duration of the migration. For more information about serialized commands, see the pipelining limitation in the Limitations section of this topic.

## What ElastiCache manages and what you manage

A migration spans your infrastructure and ours.

| ElastiCache manages | You manage |
| --- | --- |
| The migration endpoint that your application connects to. | Your source cache, including its capacity, its engine version, and its availability. ElastiCache reads from your source but does not operate it. |
| The infrastructure that reads from your source cache and replicates to your target during the migration. | The source requirements on this page, including replication support, the memory |

| ElastiCache manages | You manage |
| --- | --- |
| | limit per node, the engine version range, and network reachability. |
| The target ElastiCache Serverless cache. | Your client configuration, including retry behavior, connection pooling, and not caching node IP addresses. |

A failed request can originate on either side. If you see errors during a migration, check your source cache health and capacity first. A source that is unavailable or out of memory surfaces as errors through the migration endpoint.

The ElastiCache Service Level Agreement applies to your ElastiCache Serverless cache during a migration on the same terms as at any other time.

## Limitations

The following limitations apply during a migration:

| Limitation | What to do |
| --- | --- |
| Permanent errors | Migration errors are reported as permanent . A migration that has reported an error cannot be started again, even after you fix the underlying problem. You can stop it, and your only way to end it is to complete the migration with force. For more information, see Cutover modes for online migration to ElastiCache Serverless. |
| Force complete | Force complete does not roll back. It skips the pre-cutover health check and the cluster-w ide convergence check. For more informati on, see Cutover modes for online migration to ElastiCache Serverless. |

| Limitation | What to do |
| --- | --- |
| Rollback after cutover | There is no rollback after cutover. To return to your source cache, you must move your application traffic back to it yourself. Writes that your application makes after cutover exist only on your target. For more information, see Cutover modes for online migration to ElastiCache Serverless. |
| The reader endpoint is not supported during a migration | Send your traffic to the migration endpoint and not the cache's reader endpoint. |
| One migration per cache | A cache can be the target of only one migration at a time. |
| Pipelining is not supported during the migration | If your application pipelines commands, the migration endpoint serializes them from the moment the migration reads `ready-to-sync` . A batch of 100 commands then takes 100 round trips instead of one. Latency for those batches increases in proportion to the batch depth. |
| RESP3 Pub/Sub with regular commands on the same connection | RESP3 lets a client run regular commands on a connection that has active Pub/Sub subscriptions (`SUBSCRIBE` or `SSUBSCRIBE` ). This isn't supported while a migration is in progress. During migration, use separate connections for Pub/Sub and for regular commands. Most client libraries do this by default. This restriction ends when the migration completes. |
| Total migration throughput scales with your source shard count | A source with many shards migrates faster than a source with few. |

| Limitation | What to do |
| --- | --- |
| High-throughput sources with few shards | This release is not suited to a high-throughput source with few shards. Migration throughput is provisioned per source shard. A source that concentrates high traffic into a small number of shards is most likely to be limited. If that describes your source, review the CPU headroom on source primaries prerequisite in the source cache requirements, or plan the migration for a lower-traffic period. |
| Billing after you stop the data sync | Billing does not stop when you stop the data sync. `StopMigration` leaves the migration infrastructure provisioned. `CompleteMigration` is the call that ends the migration and its charges. Your target ElastiCache Serverless cache continues to bill until you delete it. |

## Troubleshooting a failed prepare step

When the prepare step fails, the target cache's Events tab shows the reason. The following table lists the reasons and what to check.

| Reason in the event | What to check |
| --- | --- |
| "The serverless cache must have no keys before migration can begin." | Your target contains data. Run `FLUSHALL` on the target before you prepare the migration, or choose an empty cache. |
| "The serverless cache is not yet ready for migration. Please try again later." | Wait, and then prepare the migration again. |
| "Unable to connect to source cluster." | This reason covers several causes. Check each of the following: |

| Reason in the event | What to check |
| --- | --- |
| | • The source security group allows inbound TCP on the source port from the target cache's security group or subnets.<br>• Each source node binds to its private IP address and reports a reachable address in its cluster topology.<br>• The source address is a private IP address or a host name that resolves through public DNS.<br>• Protected mode is off on a source that has no password.<br>• The migration user has every permission in Permissions for the migration user. |
| An access denied message for the secret | The secret's resource policy grants `secretsmanager:GetSecretValue` to `elasticache.amazonaws.com`, the key policy grants `kms:Decrypt`, neither statement has an `aws:SourceAccount` or `aws:SourceArn` condition, and your IAM identity has `secretsmanager:GetSecretValue`, `kms:Decrypt` and `ec2:DescribeSubnets`. |
| A subnet message that names Availability Zones | Create a new target cache with a subnet in each Availability Zone that the message names, and prepare the migration again. |
| A failure after you provide TLS settings | Provide the PEM-encoded CA certificates with `verify-ca`. The event might not name the certificate authority. |

### If a key name exceeds 4 KiB

An oversized key name fails the migration during the initial data load. Recover as follows:

1. Point your application back at your source's own endpoint.
2. End the migration with force complete, which switches the migration endpoint to your target.
3. Flush the target.
4. Remove or rename every key on your source whose name exceeds 4 KiB.
5. Prepare a new migration.

For more information, see Cutover modes for online migration to ElastiCache Serverless.

### Related topics

- Online migration to ElastiCache Serverless
- Source authentication for online migration to ElastiCache Serverless
- Migrate your data to ElastiCache Serverless
- Cutover modes for online migration to ElastiCache Serverless
- Create a Valkey serverless cache

## Source authentication for online migration to ElastiCache Serverless

If your source cache requires authentication, ElastiCache needs credentials to connect to it. You store the credentials in AWS Secrets Manager and give ElastiCache the secret's ARN when you prepare the migration. ElastiCache does not accept credentials directly.

Your target cache must also accept the credentials that your application uses on your source. For more information about setting up target users, see Setting up your application's users on your target.

If your source cache accepts unauthenticated connections, you can skip the authentication configuration.

## Storing source credentials

Store your source cache credentials in a Secrets Manager secret. You can use a secret you already manage, or create one for the migration. If you create the secret from the ElastiCache console as part of setting up the migration, ElastiCache configures its access for you.

In the console, you choose one of three authentication methods:

Use an existing secret

Choose an existing secret from Secrets Manager.

Create and use a new secret

ElastiCache will create a secret to store your specified credentials.

None

Source doesn't require authentication.

When you create and use a new secret, you also choose the AWS KMS key that encrypts it: "Choose an existing customer managed KMS key for the new secret. The KMS key you choose must be configured to grant ElastiCache decrypt access."

The secret's contents depend on how your source authenticates.

If your source authenticates users by name, store a user name and a password in the following format:

```
{"username":"migration_user","password":"your-password"}
```

Where:

- `migration_user` is the name of the user that ElastiCache connects as on your source. It must be 1-120 characters, with no spaces or other whitespace.
- `your-password` is that user's password. It must be 16-128 characters and cannot contain the characters `, " / @`.

If your source uses a single password with no user names, store the password on its own:

```
{"password":"your-password"}
```

Where:

- `your-password` is the password for your source cache. It must be 16-128 characters and can contain only letters, digits, and the characters `! & # $ ^ < > -`.

The prepare step validates the secret's format, whether you created the secret or ElastiCache created it for you. An empty secret, a secret that is not valid JSON, a secret with no `password` field, or a user name or password that does not meet the requirements above each fail with a specific message. If your source password does not meet them, change it on your source before you prepare the migration.

**Granting ElastiCache access to the secret**

ElastiCache reads the secret with its own service credentials, so the secret needs a resource policy. Grant `secretsmanager:GetSecretValue` to the `elasticache.amazonaws.com` service principal.

The following permissions are required:

| Who | Permission | Where you grant it |
|---|---|---|
| The `elasticache.amazonaws.com` service principal | `secretsmanager:GetSecretValue` | The secret's resource policy |
| The `elasticache.amazonaws.com` service principal | `kms:Decrypt` | The key policy |
| Your IAM identity | `secretsmanager:GetSecretValue` , `kms:Decrypt` , `ec2:DescribeSubnets` | Your IAM identity's permissions policy |

The following resource policy grants ElastiCache read access to the secret.

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

```
      "Principal": { "Service": "elasticache.amazonaws.com" },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*"
    }
  ]
}
```

If the secret is encrypted with a customer managed AWS KMS key, grant that service principal permission to use the key in the key policy.

Add the following statement to the key policy of the customer managed key that encrypts the secret.

```
{
  "Effect": "Allow",
  "Principal": { "Service": "elasticache.amazonaws.com" },
  "Action": "kms:Decrypt",
  "Resource": "*"
}
```

**Your secret cannot use the default AWS managed key for Secrets Manager.** That key cannot be granted to the ElastiCache service principal, so encrypt the secret with a customer managed key.

If the resource policy or the key policy leaves out the service principal, ElastiCache cannot retrieve the credentials, and the prepare step fails with an access denied message.

Do not add an `aws:SourceAccount` or `aws:SourceArn` condition to these statements. A condition on either key denies the read, and the prepare step fails with an access denied message.

The IAM identity that calls `PrepareMigration` also needs `secretsmanager:GetSecretValue` on the secret, and `kms:Decrypt` on the key that encrypts it. When it calls `PrepareMigration` and `StartMigration`, it also needs `ec2:DescribeSubnets`, because ElastiCache checks the subnets of your target cache with your credentials. ElastiCache first reads the secret with your credentials to confirm that you have access, and then with its own service credentials. If either read is denied, the prepare step fails with the same access denied message, so check both your own permissions and the secret's resource and key policies.

## Permissions for the migration user

The credentials you store decide which user ElastiCache connects as. What that user is allowed to do on your source determines whether the migration can finish.

If your source uses a single password and no user names, ElastiCache connects with that password and there is nothing further to configure.

If your source authenticates users by name and controls what each one can do, create a user for the migration and grant it the following permissions. Run this command against every node in your source, or use whatever mechanism your source has for applying access control across the cluster:

```
valkey-cli ACL SETUSER migration_user on '>your-password' -@all \
  +ping +info +hello \
  +cluster|nodes +cluster|shards +cluster|slots +cluster|info \
  +acl|list +acl|dryrun \
  +psync +sync +replconf \
  +multi +exec +client|pause +client|kill +client|unpause
```

This block applies as written to every source version that supports access control lists (ACLs): Redis OSS 6.0 and later, and all supported Valkey versions.

Where:

- `-@all` removes every command, and the commands added after it are the only ones this user can run. The user cannot write to your source and cannot change your source's access control configuration.
- `+ping`, `+info`, `+hello`, and the `+cluster` commands let ElastiCache connect to your source and discover its topology.
- `+acl|list` lets ElastiCache read your source's access control configuration. It does not allow changes to it.
- `+acl|dryrun` lets ElastiCache verify the rest of these grants during the prepare step without running the commands, on Redis OSS 7.0 and later and on Valkey. Redis OSS 6.0 and 6.2 accept the grant, and ElastiCache does not use it there.
- `+psync`, `+sync`, and `+replconf` are the replication commands ElastiCache uses to synchronize your data.
- `+multi`, `+exec`, `+client|pause`, `+client|kill`, and `+client|unpause` are required. ElastiCache uses these commands to run the cutover and validates them when you prepare the migration, except `+client|unpause` on Redis OSS 6.0 and 6.2.

ElastiCache verifies these grants when you prepare the migration and again when you start the data sync, and a missing grant fails that step. The failure event might not name the missing

permission. It can report instead that ElastiCache is unable to connect to your source, so if your source is reachable, check the migration user's grants against this list. After you prepare the migration, do not change or remove the migration user or its permissions. Doing so during a migration can cause data loss or make your cache unavailable.

### Setting up your application's users on your target

Before you prepare the migration, set up authentication on your target cache to match your source. Create an ElastiCache user for each user that your application uses on your source, with the same user name, password, and permissions. Add those users to a user group, and then assign the group to your target cache. For more information, see Role-Based Access Control (RBAC) and Assigning User Groups to Serverless Caches Using the AWS CLI.

If your source uses a single password with no user names, your application authenticates as the default user. The default user in your target's user group must have that same password.

ElastiCache user passwords must be 16 to 128 printable characters and cannot contain a comma, a double quotation mark, a slash, or an at sign. Your target checks your application's credentials during the migration and after cutover, so your application's password must meet these rules. If it does not, change the password on your source and in your application before you prepare the migration.

Your application's own users must exist on both your source and your target. During a migration, the migration endpoint checks your application's credentials on your target first and then on your source. If your source and your target do not have a consistent ACL configuration, your application can encounter authentication errors during the migration. For example, a user that is missing on your target, or that has a different password there, fails with `WRONGPASS` as soon as your application authenticates through the migration endpoint.

If your source accepts connections without authentication, keep the default user on your target enabled with no password. A client that does not authenticate gets a `NOAUTH` error from the migration endpoint when the default user on your target is disabled or has a password.

### Providing the secret when you prepare a migration

Pass the secret ARN in the source authentication configuration:

```
aws elasticache prepare-migration \
    --serverless-cache-name my-cache \
```

```
    --customer-node-endpoint-list "Address='10.0.0.10',Port=6379" \
    --source-authentication-configuration \
        SecretManagerConfig={SecretArn=arn:aws:secretsmanager:region:account:secret:my-
secret}
```

Where:

- `my-cache` is the name of the ElastiCache Serverless cache you are migrating into.
- `10.0.0.10` and `6379` are the private address and port of one node in your source cache. ElastiCache discovers the rest of your topology from this node.
- `arn:aws:secretsmanager:region:account:secret:my-secret` is the ARN of the secret that holds your source credentials.

ElastiCache validates connectivity and credentials during the prepare step. If it cannot connect, the prepare step fails. ElastiCache reports a failure event on the cache that names the reason. The migration is rolled back, so the `Migration` object is removed and your cache returns to `available`. Correct the address, the network configuration, or the secret access, then try again.

### Encryption in transit

If your source cache requires TLS, set `SourceTlsMode` to `verify-ca` in the `SourceTlsConfiguration` parameter when you prepare the migration, and pass your CA certificates in `SourceTlsCertificate`.

Provide the PEM-encoded CA certificates that issued your source's server certificate. Provide them whether a public or private certificate authority issued your source's certificate, or the certificate is self-signed. If you omit them, the migration fails.

To find the cause of that failure, read the events on your target cache. The failure might not identify the certificate authority as the reason.

The prepare step validates the certificate value. It fails if the value is not a valid PEM-encoded X.509 certificate, is longer than 5,000 characters, contains a private key, or holds a certificate that is expired or not yet valid. The 5,000 character limit counts the whole value, including the BEGIN and END lines and the line breaks, so a chain of several certificates can exceed it.

ElastiCache verifies that your source's server certificate chains to a CA certificate you provided. It does not check the host name in the certificate, so your source address can be an IP address or a host name that resolves through public DNS.

```
aws elasticache prepare-migration \
    --serverless-cache-name my-cache \
    --customer-node-endpoint-list "Address='10.0.0.10',Port=6379" \
    --source-tls-configuration SourceTlsMode=verify-
ca,SourceTlsCertificate="$(cat ca.pem)"
```

Where:

- `ca.pem` is a file that holds your PEM-encoded CA certificates. The file must contain only the PEM certificate blocks, with no other text.

If your source also requires authentication, add the `--source-authentication-configuration` parameter shown earlier on this page.

If your source accepts unencrypted connections, set `SourceTlsMode` to none. Provide a certificate only with `verify-ca`. A certificate with none, or with no TLS mode set, fails the prepare step.

### Related topics

- Prerequisites for online migration to ElastiCache Serverless
- Online migration to ElastiCache Serverless
- Migrate your data to ElastiCache Serverless
- Cutover modes for online migration to ElastiCache Serverless

## Migrate your data to ElastiCache Serverless

This page walks you through an online migration from a self-managed Valkey or Redis OSS source on Amazon EC2 to an ElastiCache Serverless for Valkey cache. You can use the ElastiCache console or the AWS CLI. You create the migration, redirect your application to the migration endpoint, start the data sync, and complete the cutover.

### Before you start

Before you begin, confirm that your source cache and your target cache meet the migration requirements. For more information, see Prerequisites for online migration to ElastiCache Serverless. If your source requires authentication or TLS, set that up first. For more information, see Source authentication for online migration to ElastiCache Serverless.

Before you migrate, test your workload on an ElastiCache Serverless cache to see how it scales and how many ElastiCache Processing Units (ECPUs) it consumes. For more information, see Scaling ElastiCache Serverless clusters.

Confirm that you have the following:

- An ElastiCache Serverless for Valkey cache that is available and empty, with the IPv4 network type and no public endpoint.
- A self-managed Valkey or Redis OSS source on Amazon EC2 that meets the source requirements.
- A Secrets Manager secret that holds your source credentials, if your source requires authentication.
- Your application's users set up on your target cache, with the same user names, passwords, and permissions as on your source, if your source requires authentication. For more information about setting up target users, see Setting up your application's users on your target.
- Your source TLS configuration, if your source requires TLS.
- At least three subnets in different Availability Zones for your target cache, each with free IP addresses.
- AWS CLI version 2.38.3 or later, if you use the AWS CLI.

## Create the migration

You can create a migration with the ElastiCache console or the AWS CLI. You can open the **Create migration** page in two ways.

- In the navigation pane, choose **Migrations**, choose **Create migration**, and then choose **Migrate to serverless**.
- On the Valkey caches page, select your target cache, or open its cache details page, and then choose **Migrate data**. This action preselects the target cache.

On the **Create migration** page, complete the following steps.

1. Under **Source details**, for **Server address**, enter your source address. You can also choose an Amazon EC2 instance from the suggestions, which fills in the address. For the source address, enter the private IP address of a source node, or a host name that resolves through public DNS. A host name that resolves only in your VPC's private DNS fails the prepare step.
2. For **Port**, enter your source port.

3. For **Authentication method**, choose one of the following options.

   **Use an existing secret**

   Choose your secret under **Existing secret**.

   **Create and use a new secret**

   Enter a user name if your engine version uses one, enter a password, and choose a customer managed AWS KMS key. ElastiCache creates the secret. The user name and password must meet the requirements in Source authentication for online migration to ElastiCache Serverless.

   **None**

   Your source does not require authentication.

4. For **TLS encryption**, choose **Verify CA** or **None**.

   If you choose **Verify CA**, under **Certificate authority** choose **Public CA** or **Private CA**, and provide the PEM-encoded CA certificates that issued your source's server certificate. Provide them for public and private authorities alike. For more information about why, see Source authentication for online migration to ElastiCache Serverless.

5. **Important**

   Your target cache must be empty when you create the migration. If it contains any keys, the prepare step fails and the target's Events tab shows "The serverless cache must have no keys before migration can begin." To empty the target, run FLUSHALL on it before you create the migration. FLUSHALL removes every key in the cache. You can have ElastiCache clear the target for you only when you start the data sync.

   Under **Target details**, for **Choose target cache**, choose your cache. Only compatible caches are listed.

6. Choose **Create**.

The console opens the migration details page. Its title reads **Migration:** followed by your cache name.

**To do this with the AWS CLI**

- Prepare the migration. For the authentication and TLS options, see Source authentication for online migration to ElastiCache Serverless.

  ```
  aws elasticache prepare-migration \
      --serverless-cache-name my-cache \
      --customer-node-endpoint-list "Address='10.0.0.10',Port=6379"
  ```

  Where:

  - `my-cache` is the name of your target cache.
  - `10.0.0.10` and `6379` are the private address and port of one node in your source.

To check the migration state, see Migration states.

## The migration details page

The migration details page shows the state and controls for your migration. The **Migration overview** shows **Target cache**, **Current state**, **Migration endpoint**, and **Cutover method**. The page has a **Monitor** tab and an **Events** tab. It also has **Stop**, **Force complete**, and **Start migration** buttons.

While ElastiCache prepares the migration, **Current state** reads **Preparing migration**, and the **Migration endpoint** field is hidden.

How long the prepare step takes depends on the size of your source. While **Current state** reads **Preparing migration**, the step is still running.

## Redirect your application

> **Warning**
>
> Point your application at the migration endpoint only when Current state reads Ready to sync. While the migration is preparing, the endpoint does not accept connections. Before you prepare a migration, and after a prepare step that fails, the endpoint serves your target cache, which is empty.

When **Current state** reads **Ready to sync**, the **Migration endpoint** field shows the address.

1. To copy the address, choose **Copy migration endpoint**.
2. Update your application to connect to the migration endpoint with TLS, and allow your application in your target cache's security group. For more information, see the section called "Client configuration".
3. Read a key that exists on your source through the migration endpoint, and confirm that it returns the value from your source.

Starting the data sync first does not affect the data on your target. Your traffic is routed to your source until cutover.

> **Important**
>
> If you use Auto-cutover, redirect your application before you start the data sync, because ElastiCache cuts over at the next fully synced window.

> **Note**
>
> A client that still connects straight to your source after cutover writes to a cache that no longer replicates to your target.

**To do this with the AWS CLI**

- Get the migration endpoint and point your application at it. Connect with TLS. Before you move all of your traffic, read a key that exists on your source through the migration endpoint. When the read returns the value from your source, the endpoint is routing to your source.

```
aws elasticache describe-serverless-caches \
    --serverless-cache-name my-cache \
    --query "ServerlessCaches[0].Migration.MigrationEndpoint"
```

To check the migration state, see Migration states.

## Start the data sync

When you start the data sync, ElastiCache begins replicating data from your source to your target cache. Your application continues to be served throughout. For more information about cutover modes, see Cutover modes for online migration to ElastiCache Serverless.

1. On the migration details page, choose **Start migration**.
2. In the **Start migration** dialog box, for **Cutover mode**, choose **Manual cutover** or **Auto-cutover**. For more information, see Cutover modes for online migration to ElastiCache Serverless.
3. **Warning**

   Clearing the target cannot be undone.

   Under **Target cache data**, to clear a target that contains data, select **Authorize automatic clearing of the target cache if it contains data**. Your target contains data if you are starting the data sync again after you stopped it. It can also contain data if an earlier start failed. If your target contains data and you do not select this option, the data sync fails because the target is not empty, and the migration returns to **Ready to sync**.
4. Choose **Start migration**.

**Current state** moves to **Initializing sync** and then to **Data sync in progress**. The migration stays in **Initializing sync** until replication begins.

**To do this with the AWS CLI**

- Start the data sync. To have ElastiCache cut over for you, add `--auto-cutover`, and redirect your application before you run this command.

```
aws elasticache start-migration \
    --serverless-cache-name my-cache
```

To check the migration state, see Migration states.

## Complete the cutover

How the cutover completes depends on the cutover mode you chose.

**Manual cutover**

When **Current state** reads **Data sync in progress** and you are ready, choose **Complete migration**. The console shows "Complete migration triggered for *name*." **Complete migration** is available only while the data sync is in progress.

**Auto-cutover**

ElastiCache cuts over at the next fully synced window. **Complete migration** is disabled in this mode.

During cutover, **Current state** reads **Cutover in progress**. **Stop** and **Force complete** are disabled during cutover.

For most of the **Cutover in progress** state, ElastiCache is removing the migration infrastructure, and your traffic is already on your target cache.

With Auto-cutover, the console opens the cache details page when the migration finishes. This page shows a **Migration completed successfully** banner until you dismiss it. The **Events** tab lists the migration events.

**To do this with the AWS CLI**

- Complete the migration. Skip this step if you chose automatic cutover.

```
aws elasticache complete-migration \
    --serverless-cache-name my-cache
```

To check the migration state, see Migration states.

After the migration completes, verify your data and clean up. For the steps, see After cutover.

## If you need to stop

To stop the data sync, choose **Stop** on the migration details page. **Stop** is available only while **Current state** reads **Data sync in progress**. While the data sync stops, **Current state** reads **Stopping**. For more information about what stopping does, see Cutover modes for online migration to ElastiCache Serverless.

**Force complete**

Force complete ends the migration even if your target is not in sync with your source. Your migration endpoint then serves from your target, which can have incomplete data. To return to your source, move your application traffic back to it yourself before you force complete.

To end a migration that has reported an error, use **Force complete**. It is available when **Current state** reads **Ready to sync** or **Data sync in progress**. The console shows "Force complete triggered for *name*." For more information, see Cutover modes for online migration to ElastiCache Serverless.

**To do this with the AWS CLI**

1. To stop the data sync, run the following command.

```
aws elasticache stop-migration \
    --serverless-cache-name my-cache
```

   Where:

   - `my-cache` is the name of your ElastiCache Serverless cache.

2. To end a migration that has reported an error instead, force complete it.

```
aws elasticache complete-migration \
    --serverless-cache-name my-cache \
    --force
```

   Where:

   - `my-cache` is the name of your ElastiCache Serverless cache.

To check the migration state, see Migration states.

For the parameters of each command, see prepare-migration, start-migration, stop-migration, and complete-migration in the *AWS CLI Command Reference*.

For more information about migrating with the AWS CLI, see the AWS CLI examples on Source authentication for online migration to ElastiCache Serverless and Cutover modes for online migration to ElastiCache Serverless.

For the API operations, see PrepareMigration, StartMigration, CompleteMigration, and StopMigration in the *Amazon ElastiCache API Reference*.

## Cutover modes for online migration to ElastiCache Serverless

Cutover is the point at which your migration endpoint stops serving from your source cache and starts serving from your ElastiCache Serverless cache. Your application does not change its connection. The endpoint stays the same.

You choose how the cutover is triggered. Manual cutover is the default.

### The Start migration dialog box

You start the data sync from the Start migration dialog box in the console. It presents the following:

- The Cutover mode control, with the values Manual cutover and Auto-cutover. With manual cutover, you call `CompleteMigration` when you are ready. With automatic cutover, ElastiCache completes the cutover for you at the next fully synced window.
- An option to clear data from the target cache. Use it when you start the data sync again after you stopped it or after an earlier start failed. Without it, the data sync fails if your target contains data. For more information, see Prerequisites for online migration to ElastiCache Serverless.
- A reminder to point your application at the migration endpoint, with a button that copies it.

To start the data sync with the AWS CLI, run the following command:

```
aws elasticache start-migration \
    --serverless-cache-name my-cache \
    --auto-cutover
```

Where:

- `my-cache` is the name of your ElastiCache Serverless cache.
- `--auto-cutover` sets automatic cutover. Omit it for manual cutover, which is the default.

- `--clear-target-cache-data` clears data from your target cache before the data sync starts. Add it when you start the data sync again after you stopped it or after an earlier start failed. Clearing the target cannot be undone.

### Before you cut over

There is no way to poll migration health on demand. Prepare the migration close to the cutover window you have planned. Watch the cache's events between starting the data sync and completing the migration. If the migration is unhealthy when you call `CompleteMigration`, the call is rejected immediately. The events carry the reason.

### Manual cutover

You decide when to complete the migration. ElastiCache keeps replicating and reports the migration as `syncing` until you act. This lets you choose when to switch, for example during a period of low write traffic.

When you are ready, complete the migration:

```
aws elasticache complete-migration \
    --serverless-cache-name my-cache
```

Where:

- `my-cache` is the name of your ElastiCache Serverless cache.

ElastiCache then waits for the next fully synced window before it cuts over. Heavy write traffic on your source can increase replication lag and delay that window. Reduce writes to help the cutover happen sooner. There can be an interval between your request and the cutover.

**If a fully synced window is not found, the cutover does not complete.** ElastiCache reports an event on the cache that a suitable window to finalize the migration was not found. Reduce the write traffic on your source and try again. If your target is already in sync with your source, you can instead complete the migration with force.

### Automatic cutover

ElastiCache completes the cutover for you at the next fully synced window. Opt in when you start the data sync. This is the simpler path when you do not need a validation window before switching.

If ElastiCache cannot find a fully synced window, automatic cutover times out. ElastiCache reports an event on the cache that automatic cutover timed out. You then call `CompleteMigration` yourself after reducing write traffic. If your target is already in sync with your source, you can call `CompleteMigration` with force instead.

Manual and automatic cutover produce the same result. Choose based on how much control you want over the timing.

The following table compares the two cutover modes.

| Behavior | Manual cutover | Automatic cutover |
|---|---|---|
| How it starts | You call `CompleteMigration` | ElastiCache starts it at the next fully synced window, if you opted in when you started the data sync |
| Trigger the cutover yourself | Yes, call `CompleteMigration` | Only with force. While the automatic cutover is pending, `CompleteMigration` without force is rejected. |
| Default | Yes | No, you opt in |
| If a fully synced window is not found | The cutover does not complete, and ElastiCache reports that a suitable window to finalize the migration was not found | Automatic cutover times out, and ElastiCache reports that it timed out |
| Validation window before cutover | You choose when to switch, so you can validate your target first | None, the cutover happens at the next fully synced window |

## Stopping the data sync

You can stop the data sync while it is in progress:

```
aws elasticache stop-migration \
```

```
    --serverless-cache-name my-cache
```

Where:

- `my-cache` is the name of your ElastiCache Serverless cache.

ElastiCache stops replicating and your application continues to be served by your source cache. Nothing about your source cache changes. The migration returns to `ready-to-sync`. If you chose automatic cutover, stopping also cancels the pending cutover, unless the cutover has already begun.

> **Important**
>
> Billing does not stop when you stop the data sync. `StopMigration` leaves the migration infrastructure provisioned. `CompleteMigration` is the call that ends the migration and its charges.

Stopping has a second use besides abandoning a migration. You might stop the data sync to free capacity on your source cache for a period, then start it again when it suits you.

**Whether you can start the data sync again depends on why it stopped.**

- If you stopped it yourself and the migration has reported no errors, the migration stays open and you can start the data sync again.

  Data that was already synced stays on your target after you stop. To start the data sync again, choose **Authorize automatic clearing of the target cache if it contains data** (`ClearTargetCacheData`). Otherwise, the data sync fails because the target is not empty.
- If the migration has reported an error, `StopMigration` still succeeds and reports that the data sync stopped, but a later `StartMigration` is rejected. This applies even to a problem you have since fixed, such as a network path that dropped or a credential that changed. The only way to end the migration is then to complete it with force.

The confirmation you get from `StopMigration` is the same either way. It does not tell you whether a later start will be accepted.

Check the migration's events before you plan to start the data sync again.

**Once cutover has begun, stopping no longer applies.** While the migration is in `cutting-over`, `StopMigration` fails with `InvalidServerlessCacheStateFault`. With automatic cutover, ElastiCache accepts a `StopMigration` call made during `syncing`. If cutover begins before the stop takes effect, the cutover continues, and the completion event reports that the stop was ignored.

## What your application sees at cutover

Cutover is not instantaneous, and it is applied per shard. Your application can see the following while cutover runs:

| What your application sees | Why | What to do |
|---|---|---|
| Reads and writes keep flowing, and tail latency rises briefly. | Traffic flows continuously throughout cutover. Tail latency rises for roughly 2 to 10 seconds, depending on your workload. | Nothing. Your application keeps being served. |
| A connection with a write in flight is closed. | If your client sent a write and did not get a definitiv e reply, ElastiCache closes the connection instead of replaying the write. | Treat it the same way you would treat a connection that died with an in-flight write outside a migration. |
| Blocking commands and pub/ sub subscribers lose their connection. | Blocking commands such as BLP0P and pub/sub subscribe rs lose their connection at cutover. They must reconnect , and subscribers must re-subscribe. | Reconnect, and re-subscribe your subscribers. |
| Clients connected directly to your source are disconnected. | ElastiCache ends your source's client connections as part of the cutover. This disconnects anything you have attached straight to your source, such | Nothing for your applicati on, which keeps using the migration endpoint. |

| What your application sees | Why | What to do |
| --- | --- | --- |
| | as a monitoring agent or a second application. | |

## Force completing a migration

Force completing skips the pre-cutover health check and the cluster-wide convergence check. Each shard still runs the full cutover process and tries to cut over with consistent data.

**Incomplete data after force complete.** Force can leave your target with incomplete data. Your target can be left with incomplete or inconsistent data only if it was not in sync with your source when you forced. There is no automated rollback. After cutover, your ElastiCache Serverless cache is authoritative. To return to your source cache, you must move your application traffic back to it yourself.

Use force complete in two cases:

- Abandoning a migration.
- Ending a migration that has reported an error, because force complete is then the only way to end it.

While the migration is `ready-to-sync`, `CompleteMigration` requires force. If you force complete before the data sync starts, your target has no data. To abandon a migration, first move your application traffic back to your source, and then force complete.

Do not use force complete to finish a healthy migration that is still syncing.

To force complete a migration, run the following command:

```
aws elasticache complete-migration \
    --serverless-cache-name my-cache \
    --force
```

Where:

- `my-cache` is the name of your ElastiCache Serverless cache.

## After cutover

Your migration endpoint now serves from your ElastiCache Serverless cache, and your application needs no change. Your source cache is no longer part of the path. You can decommission it on your own schedule after you confirm that your application is healthy on the target.

Before you decommission your source, confirm that your data arrived on your target.

**To verify your data**

1. Wait a few minutes after the completion event before you compare. Right after completion, `CurrItems` and `DBSIZE` can briefly read higher than the final count.
2. Compare the `CurrItems` metric of your target with the number of keys on your source.
3. Read a few keys that your application wrote during the migration.

> **Note**
>
> Your target can hold more keys than your source, because after cutover your application writes only to your target.

After the migration completes, remove what you set up only for the migration:

**To clean up**

1. Remove the inbound rule on your source security group.
2. Remove any test client you started.
3. Remove the secret and the customer managed key if nothing else uses them.

Completing the migration ends the migration charges. Your ElastiCache Serverless cache continues to bill at the standard data storage and ECPU rates, as it would at any other time.

## Related topics

- Online migration to ElastiCache Serverless
- Prerequisites for online migration to ElastiCache Serverless
- Source authentication for online migration to ElastiCache Serverless

- Migrate your data to ElastiCache Serverless

# Online migration to ElastiCache node-based clusters

With online migration, you can migrate your data from self-managed Valkey or Redis OSS on Amazon EC2 to a node-based Amazon ElastiCache cluster.

This refers to migration from a self-managed instance to the ElastiCache service. For information on upgrading from Redis OSS to Valkey on ElastiCache see Upgrading engine versions including cross engine upgrades.

**Note**

This page covers online migration to node-based clusters. To migrate to an ElastiCache Serverless cache, see Online migration to ElastiCache Serverless. Online migration to node-based clusters doesn't support clusters that use a data tiering node type.

## Overview

Migrating your data from Valkey or Redis OSS running on Amazon EC2 to Amazon ElastiCache requires an existing or newly created Amazon ElastiCache deployment. The deployment must have a configuration that is ready for migration. It also should be in line with the configuration that you want, including attributes such as node type, number of shards, and number of replicas.

Online migration is designed for data migration from self-managed Valkey or Redis OSS on Amazon EC2 to ElastiCache, and not for moving data between ElastiCache clusters.

**Important**

We strongly recommend you read the following sections in their entirety before beginning the online migration process.

The migration begins when you call the `StartMigration` API operation or AWS CLI command. When migrating Valkey or Redis OSS cluster-mode disabled clusters, the migration process makes the primary node of the ElastiCache Valkey or Redis OSS cluster a replica of your source Valkey or Redis OSS primary. When migrating Valkey or Redis OSS cluster-mode enabled clusters, the

migration process makes the primary node of each ElastiCache shard a replica of your source cluster's corresponding shard owning the same slots.

After the client-side changes are ready, call the `CompleteMigration` API operation. This API operation promotes your ElastiCache deployment to your primary Valkey or Redis OSS deployment with primary and replica nodes (as applicable). Now you can redirect your client application to start writing data to ElastiCache. Throughout the migration, you can check the status of replication by running the valkey-cli INFO command on your source nodes and on the ElastiCache primary nodes.

## Migration steps

The following topics outline the process for migrating your data:

- Preparing your source and target for migration
- Testing the data migration
- Starting migration
- Verifying the data migration progress
- Completing the data migration
- Performing online data migration using the Console

## Preparing your source and target for migration

With these steps you can prepare to migrate your data from a self-managed Valkey or Redis OSS source on Amazon EC2 to a node-based ElastiCache cluster.

You must ensure that all four of the prerequisites mentioned following are satisfied before you start the migration from ElastiCache console, API or AWS CLI.

**To prepare your source and target Valkey or Redis OSS nodes for migration**

1. Identify the target ElastiCache deployment and make sure that you can migrate data to it.

   An existing or newly created ElastiCache deployment should meet the following requirements for migration:

   - It is using Valkey, or Redis OSS 5.0.5 or higher for cluster mode disabled or 5.0.6 or higher for cluster mode enabled.
   - It doesn't have encryption in-transit enabled.

- It has Multi-AZ enabled.
- It has sufficient memory available to fit the data from your Valkey or Redis OSS cluster. To configure the right reserved memory settings, see Managing reserved memory for Valkey and Redis OSS.
- For cluster-mode disabled, you can migrate directly from any supported Valkey or Redis OSS version to Valkey or Redis OSS version 5.0.5 onward, using the CLI or console. For cluster mode enabled, you can migrate directly from any cluster-mode enabled Valkey or Redis OSS version to Valkey or Redis OSS version 5.0.6 onward, using the CLI or console.
- Number of shards in source and target match.
- It is not part of a global datastore.
- It has data tiering disabled.

2. Make sure that the configurations of your Valkey or Redis OSS and the ElastiCache deployment are compatible.

   At a minimum, all the following in the target ElastiCache deployment should be compatible with your Valkey or Redis OSS configuration for replication:

   - Your source Valkey or Redis OSS cluster should not have AUTH enabled.
   - The config `protected-mode` should be set to `no`.
   - If you have `bind` configuration in your Valkey or Redis OSS config, then it should be updated to allow requests from ElastiCache nodes.
   - The number of logical databases should be the same on the ElastiCache node and your Valkey or Redis OSS cluster. This value is set using `databases` in the Valkey or Redis OSS config.
   - Valkey or Redis OSS commands that perform data modification should not be renamed to allow replication of the data to succeed. For example, `sync`, `psync`, `info`, `config`, `command`, and `cluster`.
   - To replicate the data from your Valkey or Redis OSS cluster to ElastiCache, make sure that there is sufficient CPU and memory to handle this additional load. This load comes from the RDB file created by your Valkey or Redis OSS cluster and transferred over the network to ElastiCache node.
   - All Valkey or Redis OSS instances in the source cluster should be running on the same port.

3. Make sure that your instances can connect with ElastiCache by doing the following:

   - Ensure that each instance's IP address is private.

- Assign or create the ElastiCache deployment in the same virtual private cloud (VPC) as your Valkey or Redis OSS on your instance (recommended).
- If the VPCs are different, set up VPC peering to allow access between the nodes. For more information on VPC peering, see Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC.
- The security group attached to your Valkey or Redis OSS instances should allow inbound traffic from ElastiCache nodes.

4. Make sure that your application can direct traffic to ElastiCache nodes after migration of data is complete. For more information, see Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC.

## Testing the data migration

After all prerequisites are complete, you can validate the migration setup for your node-based ElastiCache cluster using the AWS Management Console, ElastiCache API, or AWS CLI. The following example shows using the CLI.

Test migration by calling the `test-migration` command with the following parameters:

- `--replication-group-id` – The ID of the replication group to which data is to be migrated.
- `--customer-node-endpoint-list` – List of endpoints from which data should be migrated. List should have only one element.

The following is an example using the CLI.

```
aws elasticache test-migration --replication-group-id test-cluster --customer-node-
endpoint-list "Address='10.0.0.241',Port=6379"
```

ElastiCache will validate migration setup without any actual data migration.

## Starting migration

After all prerequisites are complete, you can begin migrating data to your node-based ElastiCache cluster using the AWS Management Console, ElastiCache API, or AWS CLI. For cluster-mode enabled, if slot migration differs, a resharding will be performed before live migration. The following example shows using the CLI.

> **Note**
>
> We recommend that you validate your setup with the `TestMigration` API operation first. This step is optional.

> **Important**
>
> When the migration starts, ElastiCache deletes all existing data in the target cluster and replaces it with data from your source.

Start migration by calling the `start-migration` command with the following parameters:

- `--replication-group-id` – Identifier of the target ElastiCache replication group
- `--customer-node-endpoint-list` – A list of endpoints with either DNS or IP addresses and the port where your source Valkey or Redis OSS cluster is running. The list can only take one element for both cluster-mode disabled and cluster-mode enabled. If you have enabled chained replication, the endpoint can point to a replica instead of the primary node in your Valkey or Redis OSS cluster.

The following is an example using the CLI.

```
aws elasticache start-migration --replication-group-id test-cluster --customer-node-
endpoint-list "Address='10.0.0.241',Port=6379"
```

As you run this command, the ElastiCache primary node (in each shard) configures itself to become a replica of your Valkey or Redis OSS instance (for cluster mode enabled, in the shard that owns the same slots). The status of ElastiCache cluster changes to **migrating** and data starts migrating from your Valkey or Redis OSS instance to the ElastiCache primary node. Depending on the size of the data and load on your Valkey or Redis OSS instance, the migration can take a while to complete. You can check the progress of the migration by running the valkey-cli INFO command on your Valkey instance and ElastiCache primary node.

After successful replication, all writes to your Valkey or Redis OSS instances propagate to the ElastiCache cluster. You can use ElastiCache nodes for reads. However, you can't write to the ElastiCache cluster. If an ElastiCache primary node has other replica nodes connected to it, these

replica nodes continue to replicate from the ElastiCache primary node. This way, all the data from your Valkey or Redis OSS cluster gets replicated to all the nodes in ElastiCache cluster.

If an ElastiCache primary node can't become a replica of your Valkey or Redis OSS instance, it retries several times before eventually promoting itself back to primary. The status of ElastiCache cluster then changes to **available**, and a replication group event about the failure to initiate the migration is sent. To troubleshoot such a failure, check the following:

- Look at the replication group event. Use any specific information from the event to fix the migration failure.
- If the event doesn't provide any specific information, make sure that you have followed the guidelines in Preparing your source and target for migration.
- Ensure that the routing configuration for your VPC and subnets allows traffic between ElastiCache nodes and your Valkey or Redis OSS instances.
- Ensure the security group attached to your Valkey or Redis OSS instances allows inbound traffic from ElastiCache nodes.
- Check Valkey or Redis OSS logs for your instances for more information about failures specific to replication.

## Verifying the data migration progress

After data migration to your node-based ElastiCache cluster has begun, you can do the following to track its progress:

- Verify that Valkey or Redis OSS `master_link_status` is up in the `INFO` command on ElastiCache primary node(s). You can also find this information in the ElastiCache console. Select the cluster and under **CloudWatch metrics**, observe **Primary Link Health Status**. After the value reaches 1, the data is in sync.
- You can check that the ElastiCache replica has an **online** state by running the `INFO` command on your Valkey or Redis OSS instances. Doing this also provides information about replication lag.
- Verify low client output buffer by using the CLIENT LIST command on your Valkey or Redis OSS instances.

After the data migration is complete, the data is in sync with any new writes coming to the primary node(s) of your Valkey or Redis OSS cluster.

## Completing the data migration

When you are ready to cut over to your node-based ElastiCache cluster, use the `complete-migration` CLI command with the following parameters:

- `--replication-group-id` – The identifier for the replication group.
- `--force` – A value that forces the migration to stop without ensuring that data is in sync.

The following is an example.

```
aws elasticache complete-migration --replication-group-id test-cluster
```

As you run this command, the ElastiCache primary node (in each shard) stops replicating from your Valkey or Redis OSS instance and promotes it to primary. This promotion typically completes within minutes. To confirm the promotion to primary, check for the event `Complete Migration successful for test-cluster`. At this point, you can direct your application to ElastiCache writes and reads. ElastiCache cluster status should change from **migrating** to **available**.

If the promotion to primary fails, the ElastiCache primary node continues to replicate from your Valkey or Redis OSS instance. The ElastiCache cluster continues to be in **migrating** status, and a replication group event message about the failure is sent. To troubleshoot this failure, look at the following:

- Check the replication group event. Use specific information from the event to fix the failure.
- You might get an event message about data not in sync. If so, make sure that the ElastiCache primary can replicate from your Valkey or Redis OSS instance and both are in sync. If you still want to stop the migration, you can run the preceding command with the `--force` option.
- You might get an event message if one of the ElastiCache nodes is undergoing a replacement. You can retry completing the migration after the replacement is complete.

## Performing online data migration using the Console

You can use the AWS Management Console to migrate your data from a self-managed Valkey or Redis OSS source on Amazon EC2 to a node-based ElastiCache cluster.

**To perform online data migration using the console**

1. Sign in to the console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. Either create a new Valkey or Redis OSS cluster or choose an existing cluster. Make sure that your source and the cluster meet the requirements in Preparing your source and target for migration, and that the following is true:

   - The cluster is in **available** status.
3. Open the migration form in one of the following ways:

   - In the navigation pane, choose **Migrations**. Choose **Create migration**, and then choose **Migrate to node-based**. On the **Migrate data from endpoint** page, for **Target ElastiCache cluster name**, choose your cluster.
   - In the cluster list, select your cluster. For **Actions**, choose **Migrate data from endpoint**. The **Migrate data to *`cluster-name`*?** dialog box opens with your cluster as the target.

   Choosing **Start migration** on the cluster details page opens the same dialog box.
4. For **Source endpoint**, enter the IP address of your source Valkey or Redis OSS cluster. For **Source port**, enter the port where your source is available.

   > **Important**
   >
   > The IP address must be exact. If you enter the address incorrectly, the migration fails.
5. (Optional) To validate your setup without migrating any data, choose **Test migration**. For more information, see Testing the data migration.
6. 
   > **Important**
   >
   > When the migration starts, ElastiCache deletes all existing data in the target cluster and replaces it with data from your source.

   Choose **Start migration**.

   As the cluster begins migration, it changes to **Modifying** and then **Migrating** status.
7. Monitor the migration progress by choosing **Events** on the navigation pane.

At any point during the migration, you can stop it. In the cluster list, select your cluster. For **Actions**, choose **Stop data migration**. This option is available while the cluster is in **Migrating** status. In the **Stop data migration to *cluster-name*?** dialog box, choose **Stop data migration**.

Stopping the migration completes it, the same way as the `CompleteMigration` API operation. ElastiCache stops replicating from your source and promotes your ElastiCache cluster to primary. The data that already synced stays in the cluster, and writes that reach your source after that point aren't copied. If the data isn't in sync, the request can fail and the cluster stays in **Migrating** status. To stop without checking that the data is in sync, select **Force** under **Force migration stop**. For more information, see Completing the data migration. The cluster then goes to **Available** status.

# Creating and managing a node-based ElastiCache cluster

If you need fine-grained control over your ElastiCache cluster, you can create a node-based cluster. ElastiCache enables you to operate a node-based cluster by choosing the node-type, number of nodes, and node placement across AWS Availability Zones for your cluster. Since ElastiCache is a fully-managed service, it automatically manages hardware provisioning, monitoring, node replacements, and software patching for your cluster.

For information on setting up see Setting up ElastiCache. For details on managing, updating or deleting nodes or clusters, see Managing nodes in ElastiCache. For an overview of the major components of an Amazon ElastiCache deployment when you create a node-based ElastiCache cluster, see these key concepts.

**Topics**


- ElastiCache components and features
- ElastiCache terminology
- Tutorial: How to create a node-based ElastiCache cluster
- Deleting a cluster
- Other ElastiCache tutorials and videos
- Managing nodes in ElastiCache
- Managing clusters in ElastiCache
- Comparing node-based Valkey, Memcached, and Redis OSS clusters
- Choosing regions and availability zones for ElastiCache


# ElastiCache components and features

Following, you can find an overview of the major components of an Amazon ElastiCache deployment.

**Topics**


- ElastiCache nodes
- ElastiCache shards
- ElastiCache clusters

- ElastiCache replication
- ElastiCache endpoints
- ElastiCache parameter groups
- ElastiCache security
- ElastiCache subnet groups
- ElastiCache backups
- ElastiCache events

## ElastiCache nodes

A *node* is the smallest building block of an ElastiCache deployment. A node can exist in isolation from or in some relationship to other nodes.

A node is a fixed-size chunk of secure, network-attached RAM. Each node runs an instance of the engine and version that was chosen when you created your cluster. If necessary, you can scale the nodes in a cluster up or down to a different instance type. For more information, see Scaling ElastiCache.

Every node within a cluster is the same instance type and runs the same cache engine. Each cache node has its own Domain Name Service (DNS) name and port. Multiple types of cache nodes are supported, each with varying amounts of associated memory. For a list of supported node instance types, see Supported node types.

You can purchase nodes on a pay-as-you-go basis, where you only pay for your use of a node. Or you can purchase reserved nodes at a much-reduced hourly rate. If your usage rate is high, purchasing reserved nodes can save you money. Suppose that your cluster is almost always in use, and you occasionally add nodes to handle use spikes. In this case, you can purchase a number of reserved nodes to run most of the time. You can then purchase pay-as-you-go nodes for the times you occasionally need to add nodes. For more information on reserved nodes, see Reserved nodes.

For more information on nodes, see Managing nodes in ElastiCache.

## ElastiCache shards

A Valkey or Redis OSS *shard* (called a *node group* in the API and CLI) is a grouping of one to six related nodes. A Valkey or Redis OSS cluster with cluster mode enabled always has at least one shard.

Sharding is a method of database partitioning that separates large databases into smaller, faster, and more easily managed parts called data shards. This can increase database efficiency by distributing operations across multiple separate sections. Using shards can offer many benefits including improved performance, scalability, and cost efficiency.

Valkey and Redis OSS clusters with cluster mode enabled can have up to 500 shards, with your data partitioned across the shards. The node or shard limit can be increased to a maximum of 500 per cluster if the Valkey or Redis OSS engine version is 5.0.6 or higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group. For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

A *multiple node shard* implements replication by having one read/write primary node and 1–5 replica nodes. For more information, see High availability using replication groups.

For more information on shards, see Working with shards in ElastiCache.

## ElastiCache clusters

A *cluster* is a logical grouping of one or more nodes. Data is partitioned across the nodes in a Memcached cluster, and across the shards in a Valkey or Redis OSS cluster that has cluster mode enabled.

Many ElastiCache operations are targeted at clusters:

- Creating a cluster
- Modifying a cluster
- Taking snapshots of a cluster (all versions of Redis)
- Deleting a cluster
- Viewing the elements in a cluster
- Adding or removing cost allocation tags to and from a cluster

For more detailed information, see the following related topics:

- Managing clusters in ElastiCache and Managing nodes in ElastiCache

  Information about clusters, nodes, and related operations.
- AWS service limits: Amazon ElastiCache

  Information about ElastiCache limits, such as the maximum number of nodes or clusters. To exceed certain of these limits, you can make a request using the Amazon ElastiCache cache node request form.
- Mitigating Failures

  Information about improving the fault tolerance of your clusters and Valkey or Redis OSS replication groups.

## Typical cluster configurations

Following are typical cluster configurations.

**Valkey or Redis OSS clusters**

Valkey or Redis OSS clusters with cluster mode disabled always contain just one shard (in the API and CLI, one node group). A Valkey or Redis OSS shard contains one to six nodes. If there is more than one node in a shard, the shard supports replication. In this case, one node is the read/write primary node and the others are read-only replica nodes.

For improved fault tolerance, we recommend having at least two nodes in a Valkey or Redis OSS cluster and enabling Multi-AZ. For more information, see Mitigating Failures.

As demand upon your Valkey or Redis OSS cluster changes, you can scale up or down. To do this, move your cluster to a different node instance type. If your application is read intensive, we recommend adding read-only replicas to the cluster. By doing this, you can spread the reads across a more appropriate number of nodes.

You can also use data-tiering. More frequently accessed data is stored in memory and less frequently accessed data is stored on disk. The advantage of using data tiering is that it decreases memory needs. For more information, see Data tiering in ElastiCache.

For Valkey 9.0 and higher, you can create your cluster with durability enabled to persist data in a Multi-AZ transactional log. With durability enabled, your data is protected even if all cache nodes fail, replicas recover independently without impacting the primary, and snapshots are created off-box without performance impact.

ElastiCache supports changing a Valkey or Redis OSS cluster's node type to a larger node type dynamically. For information on scaling up or down, see Scaling for Valkey or Redis OSS (Cluster Mode Disabled) clusters or Scaling replica nodes for Valkey or Redis OSS (Cluster Mode Disabled).

**Typical cluster configurations for Memcached**

Memcached supports up to 300 nodes per customer for each AWS Region with each cluster having 1–60 nodes. You partition your data across the nodes in a Memcached cluster.

When you run the Memcached engine, clusters can be made up of 1–60 nodes. You partition your database across the nodes. Your application reads and writes to each node's endpoint. For more information, see Auto Discovery.

For improved fault tolerance, locate your Memcached nodes in various Availability Zones (AZs) within the cluster's AWS Region. That way, a failure in one AZ has minimal impact upon your entire cluster and application. For more information, see Mitigating Failures.

As demand upon your Memcached cluster changes, you can scale out or in by adding or removing nodes, which repartitions your data across the new number of nodes. When you partition your data, we recommend using consistent hashing. For more information about consistent hashing, see Configuring your ElastiCache client for efficient load balancing (Memcached).

## ElastiCache replication

For Valkey and Redis OSS, replication is implemented by grouping from two to six nodes in a shard (in the API and CLI, called a node group). One of these nodes is the read/write primary node. All the other nodes are read-only replica nodes. Replications are only available for ElastiCache for Valkey and Redis OSS, and not for ElastiCache for Memcached.

Each replica node maintains a copy of the data from the primary node. Replica nodes use asynchronous replication mechanisms to keep synchronized with the primary node. Applications can read from any node in the cluster but can write only to primary nodes. Read replicas enhance scalability by spreading reads across multiple endpoints. Read replicas also improve fault tolerance by maintaining multiple copies of the data. Locating read replicas in multiple Availability Zones further improves fault tolerance. For more information on fault tolerance, see Mitigating Failures.

Valkey or Redis OSS clusters support one shard (in the API and CLI, called a *node group*).

Replication from the API and CLI perspective uses different terminology to maintain compatibility with previous versions, but the results are the same. The following table shows the API and CLI terms for implementing replication.

**Comparing Replication: Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled)--> Valkey or Redis OSS cluster with cluster mode enabled vs. Valkey or Redis OSS cluster with cluster mode disabled**

In the following table, you can find a comparison of the features of Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled) replication groups.

| | Valkey or Redis OSS cluster with cluster mode disabled | Valkey or Redis OSS cluster with cluster mode enabled |
|---|---|---|
| Shards (node groups) | 1 | 1–500 |
| Replicas for each shard (node group) | 0–5 | 0–5 |
| Data partitioning | No | Yes |
| Add/Delete replicas | Yes | Yes |
| Add/Delete node groups | No | Yes |
| Supports scale up | Yes | Yes |
| Supports engine upgrades | Yes | Yes |
| Promote replica to primary | Yes | Automatic |
| Multi-AZ | Optional | Required |
| Backup/Restore | Yes | Yes |
| **Notes:** | | |
| If any primary has no replicas and the primary fails, you lose all that primary's data. | | |
| You can use backup and restore to migrate to Valkey or Redis OSS (cluster mode enabled). | | |
| You can use backup and restore to resize your Valkey or Redis OSS (cluster mode enabled) cluster. | | |

All of the shards (in the API and CLI, node groups) and nodes must reside in the same AWS Region. However, you can provision the individual nodes in multiple Availability Zones within that AWS Region.

Read replicas guard against potential data loss because your data is replicated over two or more nodes—the primary and one or more read replicas. For greater reliability and faster recovery, we recommend that you create one or more read replicas in different Availability Zones.

You can also leverage Global datastores. By using the Global Datastore for Redis OSS feature, you can work with fully managed, fast, reliable, and secure replication across AWS Regions. Using this feature, you can create cross-Region read replica clusters for ElastiCache to enable low-latency reads and disaster recovery across AWS Regions. For more information, see Replication across AWS Regions using global datastores.

## ElastiCache endpoints

An *endpoint* is the unique address your application uses to connect to an ElastiCache node or cluster.

### Single node endpoints for Valkey or Redis OSS with cluster mode disabled

The endpoint for a single node Valkey or Redis OSS cluster is used to connect to the cluster for both reads and writes.

### Multi-node endpoints for Valkey or Redis OSS with cluster mode disabled

A multiple node Valkey or Redis OSS cluster with cluster mode disabled has two types of endpoints. The primary endpoint always connects to the primary node in the cluster, even if the specific node in the primary role changes. Use the primary endpoint for all writes to the cluster.

Use the Reader Endpoint to evenly split incoming connections to the endpoint between all read replicas. Use the individual Node Endpoints for read operations (In the API/CLI these are referred to as Read Endpoints).

### Valkey or Redis OSS (Cluster Mode Enabled) endpoints

A Valkey or Redis OSS cluster with cluster mode enabled has a single configuration endpoint. By connecting to the configuration endpoint, your application is able to discover the primary and read endpoints for each shard in the cluster.

For more information, see Finding connection endpoints in ElastiCache.

### ElastiCache for Memcached endpoints

Each node in a Memcached cluster has its own endpoint. The cluster also has an endpoint called the *configuration endpoint*. If you enable Auto Discovery and connect to the configuration endpoint, your application automatically *knows* each node endpoint, even after adding or removing nodes from the cluster. For more information, see Auto Discovery.

For more information, see Finding connection endpoints in ElastiCache.

## ElastiCache parameter groups

Cache parameter groups are an easy way to manage runtime settings for supported engine software. Parameters are used to control memory usage, eviction policies, item sizes, and more. An ElastiCache parameter group is a named collection of engine-specific parameters that you can apply to a cluster. By doing this, you make sure that all of the nodes in that cluster are configured in exactly the same way.

For a list of supported parameters, their default values, and which ones can be modified, see DescribeEngineDefaultParameters (CLI: describe-engine-default-parameters).

For more detailed information on ElastiCache parameter groups, see Configuring engine parameters using ElastiCache parameter groups.

## ElastiCache security

For enhanced security, ElastiCache node access is restricted to applications running on the Amazon EC2 instances that you allow. You can control the Amazon EC2 instances that can access your cluster using security groups.

By default, all new ElastiCache clusters are launched in an Amazon Virtual Private Cloud (Amazon VPC) environment. You can use *subnet groups* to grant cluster access from Amazon EC2 instances running on specific subnets.

In addition to restricting node access, ElastiCache supports TLS and in-place encryption for nodes running specified versions of ElastiCache. For more information, see the following:

- Data security in Amazon ElastiCache
- Authenticating with the Valkey and Redis OSS AUTH command

## ElastiCache subnet groups

A *subnet group* is a collection of subnets (typically private) that you can designate for your clusters running in an Amazon VPC environment.

If you create a cluster in an Amazon VPC, then you must specify a cache subnet group. ElastiCache uses that cache subnet group to choose a subnet and IP addresses within that subnet to associate with your cache nodes.

For more information about cache subnet group usage in an Amazon VPC environment, see the following:

- Amazon VPCs and ElastiCache security
- Step 3. Authorize access to the cluster
- Subnets and subnet groups

## ElastiCache backups

A *backup* is a point-in-time copy of a Valkey or Redis OSS cluster or serverless cache, or a Memcached serverless cache. Backups can be used to restore an existing cluster or to seed a new cluster. Backups consist of all the data in a cluster plus some metadata.

Depending upon the version of Valkey or Redis OSS running on your cluster, the backup process requires differing amounts of reserved memory to succeed. For more information, see the following:

- Snapshot and restore
- Performance impact of backups of node-based clusters
- Ensuring you have enough memory to make a Valkey or Redis OSS snapshot

## ElastiCache events

When important events happen on a cluster, ElastiCache sends notification to a specific Amazon SNS topic. These events can include such things as failure or success in adding a node, a security group modification, and others. By monitoring for key events, you can know the current state of your clusters and in many cases take corrective action.

For more information on ElastiCache events, see Amazon SNS monitoring of ElastiCache events.

# ElastiCache terminology

Amazon ElastiCache supports partitioning your data across up to 500 shards (called node groups in the ElastiCache API and AWS CLI).

At the same time, we began using terminology in the ElastiCache console that is used in this new functionality and common across the industry. These changes mean that at some points, the terminology used in the API and CLI might be different from the terminology used in the console. The following list identifies terms that might differ between the API and CLI and the console.

**Cache cluster or node vs. node**

There is a one-to-one relationship between a node and a cluster when there are no replica nodes. Thus, the ElastiCache console often used the terms interchangeably. The console now uses the term *node* throughout. The one exception is the **Create Cluster** button, which launches the process to create a cluster with or without replica nodes.

The ElastiCache API and AWS CLI continue to use the terms as they have in the past.

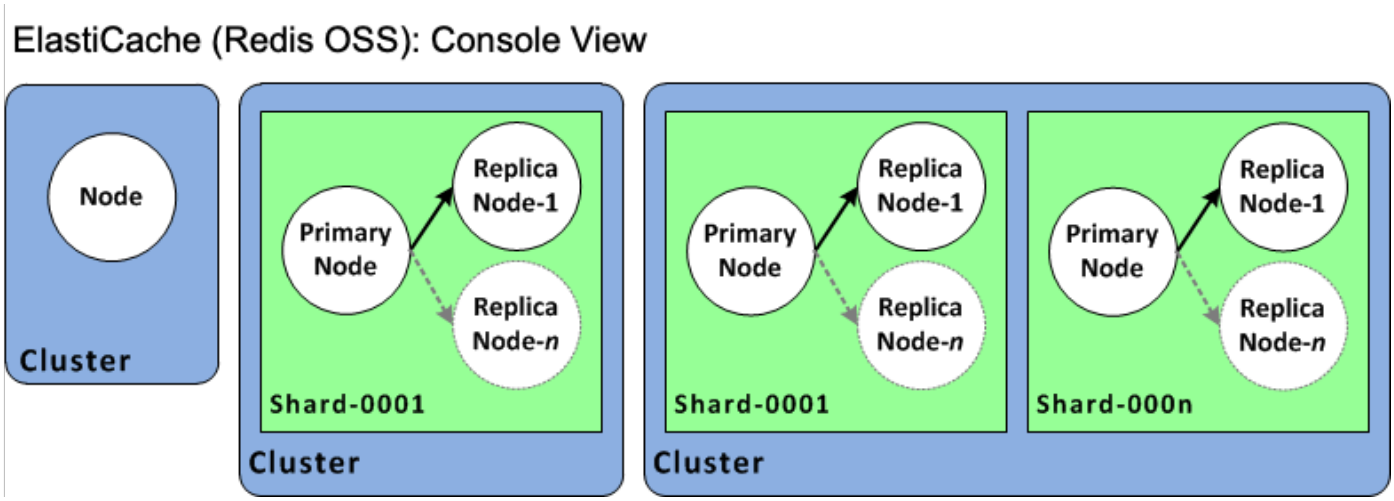
**Cluster vs. Valkey or Redis OSS replication group**

The console now uses the term *cluster* for all ElastiCache for Redis OSS clusters. The console uses the term cluster in all these circumstances:

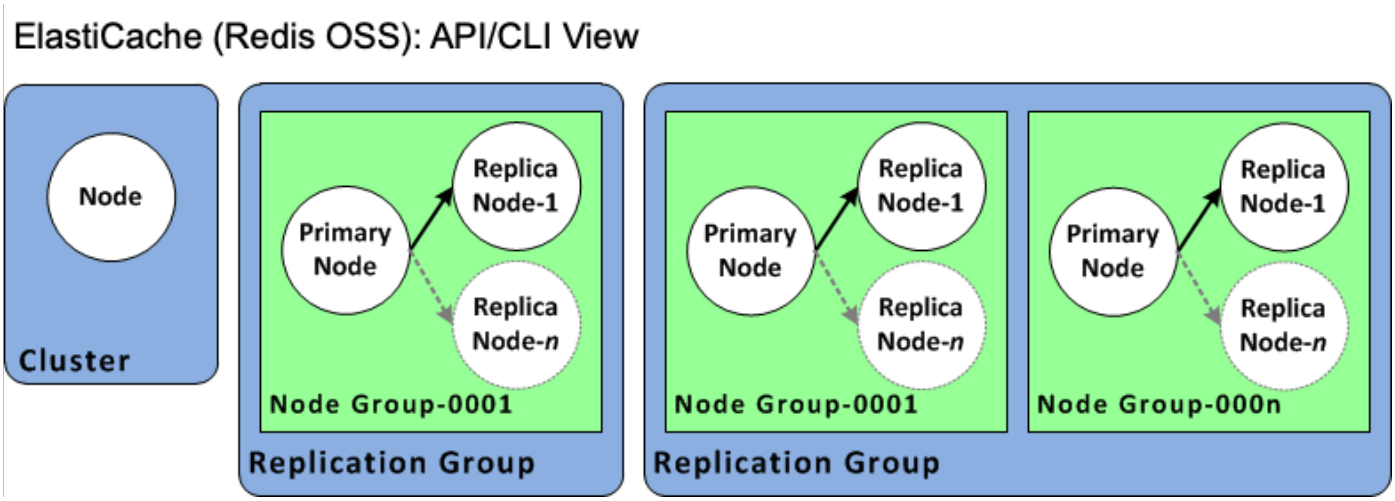
- When the cluster is a single node Valkey or Redis OSS cluster.
- When the cluster is a Valkey or Redis OSS (cluster mode disabled) cluster that supports replication within a single shard (in the API and CLI, called a *node group*).
- When the cluster is a Valkey or Redis OSS (cluster mode enabled) cluster that supports replication within 1-90 shards or up to 500 with a limit increase request. To request a limit increase, see AWS service limits and choose the limit type **Nodes per cluster per instance type**.

For more information on Valkey or Redis OSS replication groups, see High availability using replication groups.

The following diagram illustrates the various topologies of ElastiCache for Redis OSS clusters from the console's perspective.




The ElastiCache API and AWS CLI operations still distinguish single node ElastiCache for Redis OSS clusters from multi-node Valkey or Redis OSS replication groups. The following diagram illustrates the various ElastiCache for Redis OSS topologies from the ElastiCache API and AWS CLI perspective.




**Valkey or Redis OSS Replication group vs. global datastore**

A global datastore is a collection of one or more clusters that replicate to one another across Regions, whereas a Valkey or Redis OSS replication group replicates data across a cluster mode enabled cluster with multiple shards. A global datastore consists of the following:

- **Primary (active) cluster** – A primary cluster accepts writes that are replicated to all clusters within the global datastore. A primary cluster also accepts read requests.
- **Secondary (passive) cluster** – A secondary cluster only accepts read requests and replicates data updates from a primary cluster. A secondary cluster needs to be in a different AWS Region than the primary cluster.

For information on global datastores, see Replication across AWS Regions using global datastores.

# Tutorial: How to create a node-based ElastiCache cluster

Here is how to create a node-based ElastiCache cluster for Valkey, Memcached and Redis OSS.

**Topics**

- Creating a node-based ElastiCache cluster for Valkey
- Creating a node-based ElastiCache cluster for Redis OSS

## Creating a node-based ElastiCache cluster for Valkey

Following are the one-time actions you must take to create a node-based ElastiCache cluster for Valkey.

### Step 1: Create a subnet group

Before you create an ElastiCache (Valkey) cluster, you first create a subnet group. A *cache subnet group* is a collection of subnets that you may want to designate for your clusters in a VPC. When launching a cluster in a VPC, you need to select a cache subnet group. Then ElastiCache uses that cache subnet group to assign IP addresses within that subnet to each cache node in the cluster.

When you create a new subnet group, note the number of available IP addresses. If the subnet has very few free IP addresses, you might be constrained as to how many more nodes you can add to the cluster. To resolve this issue, you can assign one or more subnets to a subnet group so that you have a sufficient number of IP addresses in your cluster's Availability Zone. After that, you can add more nodes to your cluster.

For further information on setting up ElastiCache see Setting up ElastiCache.

The following procedures show you how to create a subnet group called `mysubnetgroup` (console) and the AWS CLI.

#### Creating a subnet group (Console)

The following procedure shows how to create a subnet group (console).

**To create a subnet group (Console)**

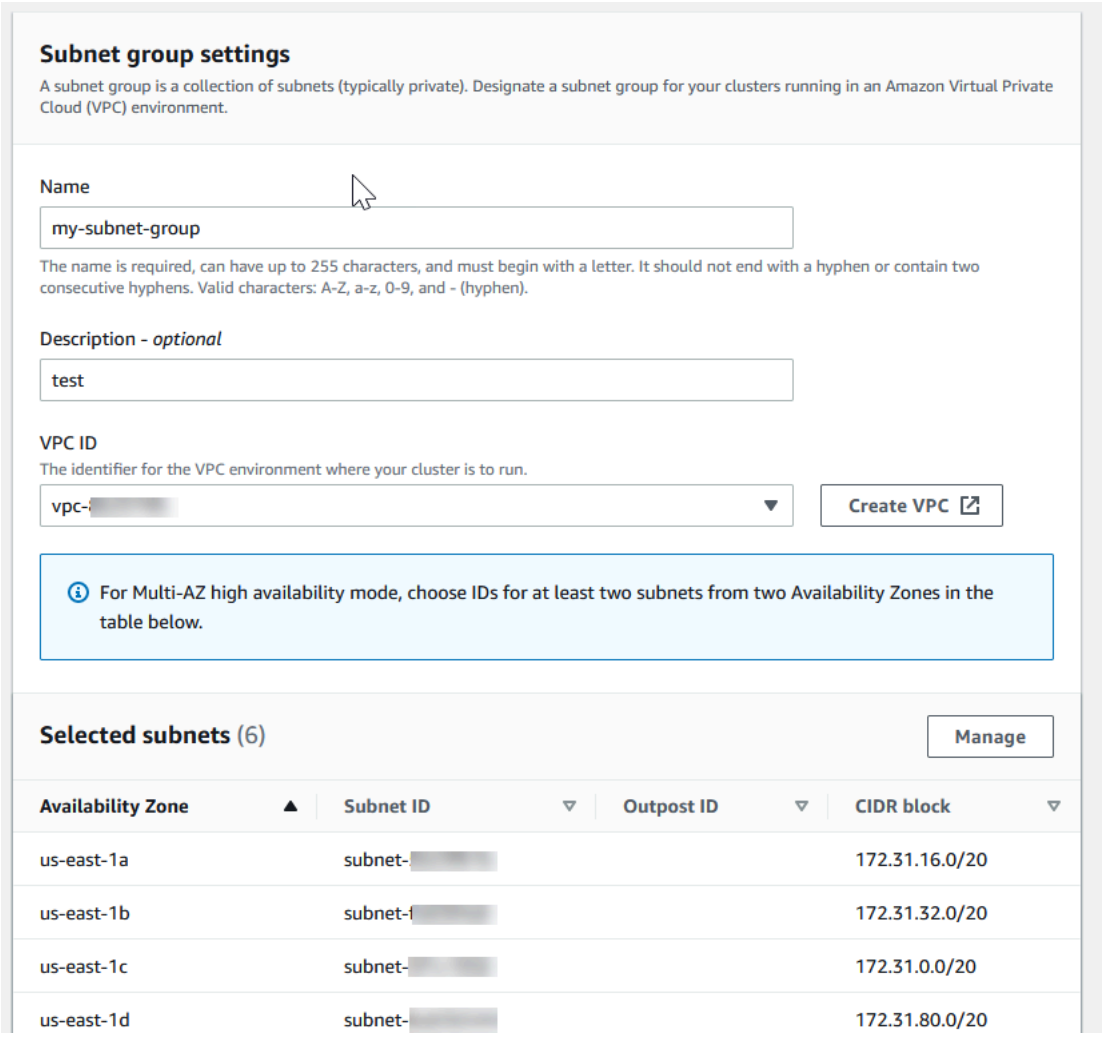
1. Sign in to the AWS Management Console, and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation list, choose **Subnet Groups**.
3. Choose **Create Subnet Group**.
4. In the **Create Subnet Group** wizard, do the following. When all the settings are as you want them, choose **Yes, Create**.

   a. In the **Name** box, type a name for your subnet group.

   b. In the **Description** box, type a description for your subnet group.

   c. In the **VPC ID** box, choose the Amazon VPC that you created.

   d. In the **Availability Zone** and **Subnet ID** lists, choose the Availability Zone or Using local zones with ElastiCache and ID of your private subnet, and then choose **Add**.



5. In the confirmation message that appears, choose **Close**.

Your new subnet group appears in the **Subnet Groups** list of the ElastiCache console. At the bottom of the window you can choose the subnet group to see details, such as all of the subnets associated with this group.

### Create a subnet group (AWS CLI)

At a command prompt, use the command `create-cache-subnet-group` to create a subnet group.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-subnet-group \
    --cache-subnet-group-name mysubnetgroup \
    --cache-subnet-group-description "Testing" \
    --subnet-ids subnet-53df9c3a
```

For Windows:

```
aws elasticache create-cache-subnet-group ^
    --cache-subnet-group-name mysubnetgroup ^
    --cache-subnet-group-description "Testing" ^
    --subnet-ids subnet-53df9c3a
```

This command should produce output similar to the following:

```
{
    "CacheSubnetGroup": {
        "VpcId": "vpc-37c3cd17",
        "CacheSubnetGroupDescription": "Testing",
        "Subnets": [
            {
                "SubnetIdentifier": "subnet-53df9c3a",
                "SubnetAvailabilityZone": {
                    "Name": "us-west-2a"
                }
            }
        ],
        "CacheSubnetGroupName": "mysubnetgroup"
```

```
    }
}
```

For more information, see the AWS CLI topic create-cache-subnet-group.

## Step 2: Create a cluster

Before creating a cluster for production use, you obviously need to consider how you will configure the cluster to meet your business needs. Those issues are addressed in the Preparing a cluster in ElastiCache section. For the purposes of this Getting Started exercise, you will create a cluster with cluster mode disabled and you can accept the default configuration values where they apply.

The cluster you create will be live, and not running in a sandbox. You will incur the standard ElastiCache usage fees for the instance until you delete it. The total charges will be minimal (typically less than a dollar) if you complete the exercise described here in one sitting and delete your cluster when you are finished. For more information about ElastiCache usage rates, see Amazon ElastiCache.

Your cluster is launched in a virtual private cloud (VPC) based on the Amazon VPC service.

**Creating a Valkey (cluster mode disabled) cluster (Console)**

**To create a Valkey (cluster mode disabled) cluster using the ElastiCache console**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region that you want to launch this cluster in.
3. Choose **Get started** from the navigation pane.
4. Choose **Create VPC** and follow the steps outlined at Creating a Virtual Private Cloud (VPC).
5. On the ElastiCache dashboard page, choose **Valkey cache** or **Redis OSS cache** and then choose **Create Valkey cache** or **Create Redis OSS cache**.
6. Under **Cluster settings**, do the following:
    a. Choose **Configure and create a new cluster**.
    b. For **Cluster mode**, choose **Disabled**.
    c. For **Cluster info** enter a value for **Name**.

d. (Optional) Enter a value for **Description**.

7. Under **Location**:

   AWS Cloud

   1. For **AWS Cloud**, we recommend you accept the default settings for **Multi-AZ** and **Auto-failover**. For more information, see Minimizing downtime in ElastiCache for Redis OSS with Multi-AZ.
   2. Under **Cluster settings**

      a. For **Engine version**, choose an available version.

      b. For **Port**, use the default port, 6379. If you have a reason to use a different port, enter the port number.

      c. For **Parameter group**, choose a parameter group or create a new one. Parameter groups control the runtime parameters of your cluster. For more information on parameter groups, see Valkey and Redis OSS parameters and Creating an ElastiCache parameter group.

      > **Note**
      >
      > When you select a parameter group to set the engine configuration values, that parameter group is applied to all clusters in the global datastore. On the **Parameter Groups** page, the yes/no **Global** attribute indicates whether a parameter group is part of a global datastore.

      d. For **Node type**, choose the down arrow (▼ ).
      In the **Change node type** dialog box, choose a value for **Instance family** for the node type that you want. Then choose the node type that you want to use for this cluster, and then choose **Save**.

      For more information, see Choosing your node size.

      If you choose a data tiering node type, ElastiCache automatically enables data tiering. For more information, see Data tiering in ElastiCache.

      e. For **Number of replicas**, choose the number of read replicas you want. If you enabled Multi-AZ, the number must be between 1-5.

3. Under **Connectivity**

   a. For **Network type**, choose the IP version(s) this cluster will support.

   b. For **Subnet groups**, choose the subnet that you want to apply to this cluster. ElastiCache uses that subnet group to choose a subnet and IP addresses within that subnet to associate with your nodes. ElastiCache clusters require a dual-stack subnet with both IPv4 and IPv6 addresses assigned to them to operate in dual-stack mode and an IPv6-only subnet to operate as IPv6-only.

   When creating a new subnet group, enter the **VPC ID** to which it belongs.

   For more information, see:

   - Choosing a network type in ElastiCache.
   - Create a subnet in your VPC.

   If you are Using local zones with ElastiCache, you must create or choose a subnet that is in the local zone.

   For more information, see Subnets and subnet groups.

4. For **Availability zone placements**, you have two options:

   - **No preference** – ElastiCache chooses the Availability Zone.
   - **Specify availability zones** – You specify the Availability Zone for each cluster.

   If you chose to specify the Availability Zones, for each cluster in each shard, choose the Availability Zone from the list.

   For more information, see Choosing regions and availability zones for ElastiCache.

5. Choose **Next**

6. Under **Advanced Valkey or Redis OSS settings**

   - For **Security**:

     i. To encrypt your data, you have the following options:

     - **Encryption at rest** – Enables encryption of data stored on disk. For more information, see Encryption at Rest.

> **Note**
>
> You have the option to supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key. For more information, see Using customer managed keys from AWS KMS.

- **Encryption in-transit** – Enables encryption of data on the wire. For more information, see encryption in transit. For Valkey and for Redis OSS 6.0 and above, if you enable Encryption in-transit you will be prompted to specify one of the following **Access Control** options:
  - **No Access Control** – This is the default setting. This indicates no restrictions on user access to the cluster.
  - **User Group Access Control List** – Select a user group with a defined set of users that can access the cluster. For more information, see Managing User Groups with the Console and CLI.
  - **AUTH Default User** – An authentication mechanism for Redis OSS server. For more information, see AUTH.
- **AUTH** – An authentication mechanism for Redis OSS server. For more information, see AUTH.

> **Note**
>
> RBAC is available on Valkey, and on Redis OSS version 6.0 and later. On earlier Redis OSS versions, AUTH is the sole option.

ii. For **Security groups**, choose the security groups that you want for this cluster. A *security group* acts as a firewall to control network access to your cluster. You can use the default security group for your VPC or create a new one.

For more information on security groups, see Security groups for your VPC in the *Amazon VPC User Guide.*

7. For regularly scheduled automatic backups, select **Enable automatic backups** and then enter the number of days that you want each automatic backup retained before it is automatically deleted. If you don't want regularly scheduled automatic backups, clear

the **Enable automatic backups** check box. In either case, you always have the option to create manual backups.

For more information on Redis OSS backup and restore, see Snapshot and restore.

8. (Optional) Specify a maintenance window. The *maintenance window* is the time, generally an hour in length, each week when ElastiCache schedules system maintenance for your cluster. You can allow ElastiCache to choose the day and time for your maintenance window (*No preference*), or you can choose the day, time, and duration yourself (*Specify maintenance window*). If you choose *Specify maintenance window* from the lists, choose the *Start day*, *Start time*, and *Duration* (in hours) for your maintenance window. All times are UCT times.

   For more information, see Managing ElastiCache cluster maintenance.

9. (Optional) For **Logs**:

   - Under **Log format**, choose either **Text** or **JSON**.
   - Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.
   - Under **Log destination**, choose either **Create new** and enter either your CloudWatch Logs log group name or your Firehose stream name, or choose **Select existing** and then choose either your CloudWatch Logs log group name or your Firehose stream name,

10. For **Tags**, to help you manage your clusters and other ElastiCache resources, you can assign your own metadata to each resource in the form of tags. For mor information, see Tagging your ElastiCache resources.
11. Choose **Next**.
12. Review all your entries and choices, then make any needed corrections. When you're ready, choose **Create**.

On premises

1. For **On premises**, we recommend you leave **Auto-failover** enabled. For more information, see Minimizing downtime in ElastiCache for Redis OSS with Multi-AZ
2. To finish creating the cluster, follow the steps at Using Outposts.

As soon as your cluster's status is *available*, you can grant Amazon EC2 access to it, connect to it, and begin using it. For more information, see Step 3. Authorize access to the cluster and Step 4. Connect to the cluster's node.

> **Important**
>
> Once your cluster is available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

To work with cluster mode enabled, see the following topics:

- To use the console, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console).
- To use the AWS CLI, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (AWS CLI).

## Step 3. Authorize access to the cluster

This section assumes that you are familiar with launching and connecting to Amazon EC2 instances. For more information, see the *Amazon EC2 Getting Started Guide*.

All ElastiCache clusters are designed to be accessed from an Amazon EC2 instance. The most common scenario is to access an ElastiCache cluster from an Amazon EC2 instance in the same Amazon Virtual Private Cloud (Amazon VPC), which will be the case for this exercise.

By default, network access to your cluster is limited to the account that was used to create it. Before you can connect to a cluster from an EC2 instance, you must authorize the EC2 instance to access the cluster.

The most common use case is when an application deployed on an EC2 instance needs to connect to a cluster in the same VPC. The simplest way to manage access between EC2 instances and clusters in the same VPC is to do the following:

1. Create a VPC security group for your cluster. This security group can be used to restrict access to the cluster instances. For example, you can create a custom rule for this security group that allows TCP access using the port you assigned to the cluster when you created it and an IP address you will use to access the cluster.

   The default port for Valkey or Redis OSS clusters and replication groups is 6379.

   > **Important**
   >
   > Amazon ElastiCache security groups are only applicable to clusters that are *not* running in an Amazon Virtual Private Cloud environment (VPC). If you are running in an Amazon Virtual Private Cloud, **Security Groups** is not available in the console navigation pane.
   > If you are running your ElastiCache nodes in an Amazon VPC, you control access to your clusters with Amazon VPC security groups, which are different from ElastiCache security groups. For more information about using ElastiCache in an Amazon VPC, see Amazon VPCs and ElastiCache security

2. Create a VPC security group for your EC2 instances (web and application servers). This security group can, if needed, allow access to the EC2 instance from the Internet via the VPC's routing table. For example, you can set rules on this security group to allow TCP access to the EC2 instance over port 22.

3. Create custom rules in the security group for your Cluster that allow connections from the security group you created for your EC2 instances. This would allow any member of the security group to access the clusters.

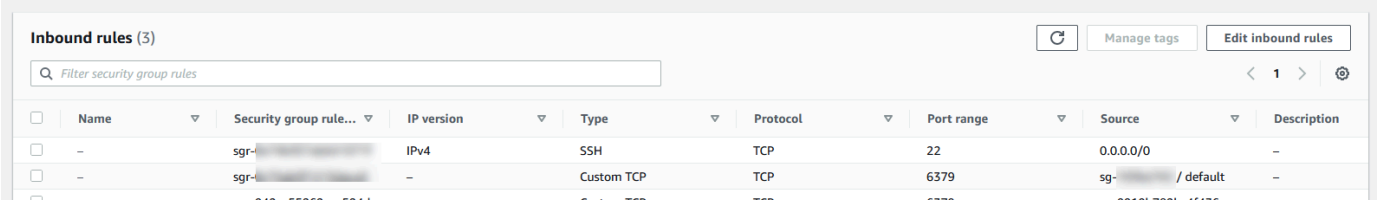
> **Note**
>
> If you are planning to use Local Zones, ensure that you have enabled them. When you create a subnet group in that local zone, your VPC is extended to that Local Zone and your VPC will treat the subnet as any subnet in any other Availability Zone. All relevant gateways and route tables will be automatically adjusted.

**To create a rule in a VPC security group that allows connections from another security group**

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc.
2. In the navigation pane, choose **Security Groups**.
3. Select or create a security group that you will use for your Cluster instances. Under **Inbound Rules**, select **Edit Inbound Rules** and then select **Add Rule**. This security group will allow access to members of another security group.
4. From **Type** choose **Custom TCP Rule**.
   a. For **Port Range**, specify the port you used when you created your cluster.

      The default port for Valkey or Redis OSS clusters and replication groups is 6379.

   b. In the **Source** box, start typing the ID of the security group. From the list select the security group you will use for your Amazon EC2 instances.
5. Choose **Save** when you finish.




Once you have enabled access, you are now ready to connect to the node, as discussed in the next section.

For information on accessing your ElastiCache cluster from a different Amazon VPC, a different AWS Region, or even your corporate network, see the following:

- Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC
- Accessing ElastiCache resources from outside AWS

## Step 4. Connect to the cluster's node

Before you continue, complete Step 3. Authorize access to the cluster.

This section assumes that you've created an Amazon EC2 instance and can connect to it. For instructions on how to do this, see the Amazon EC2 Getting Started Guide.

An Amazon EC2 instance can connect to a cluster node only if you have authorized it to do so.

### Find your node endpoints

When your cluster is in the *available* state and you've authorized access to it, you can log in to an Amazon EC2 instance and connect to the cluster. To do so, you must first determine the endpoint.

#### Finding a Valkey (cluster mode disabled) cluster's endpoints (Console)

If a Valkey (cluster mode disabled) cluster has only one node, the node's endpoint is used for both reads and writes. If the cluster has multiple nodes, there are three types of endpoints; the *primary endpoint*, the *reader endpoint* and the *node endpoints*.

The primary endpoint is a DNS name that always resolves to the primary node in the cluster. The primary endpoint is immune to changes to your cluster, such as promoting a read replica to the primary role. For write activity, we recommend that your applications connect to the primary endpoint.

A reader endpoint will evenly split incoming connections to the endpoint between all read replicas in a ElastiCache cluster. Additional factors such as when the application creates the connections or how the application (re)-uses the connections will determine the traffic distribution. Reader endpoints keep up with cluster changes in real-time as replicas are added or removed. You can place your ElastiCache cluster's multiple read replicas in different AWS Availability Zones (AZ) to ensure high availability of reader endpoints.

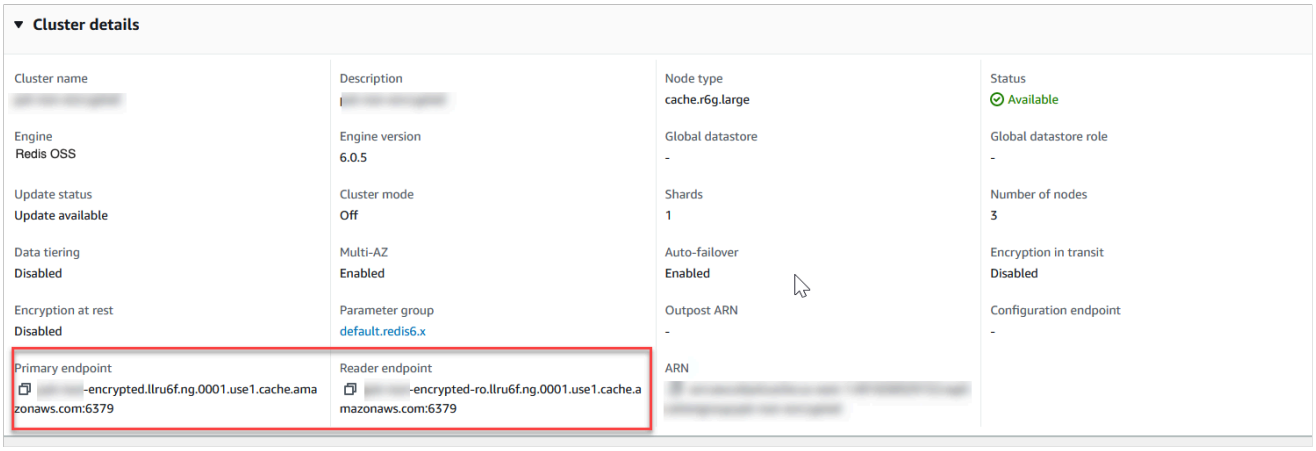
> **Note**
>
> A reader endpoint is not a load balancer. It is a DNS record that will resolve to an IP address of one of the replica nodes in a round robin fashion.

For read activity, applications can also connect to any node in the cluster. Unlike the primary endpoint, node endpoints resolve to specific endpoints. If you make a change in your cluster, such as adding or deleting a replica, you must update the node endpoints in your application.

### To find a Valkey (cluster mode disabled) cluster's endpoints

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey caches** or **Redis OSS caches**.

   The clusters screen will appear with a list that will include any existing Valkey or Redis OSS serverless caches, Valkey (cluster mode disabled) and Valkey (cluster mode enabled) clusters. Choose the cluster you created in the Creating a Valkey (cluster mode disabled) cluster (Console) section.
3. To find the cluster's Primary and/or Reader endpoints, choose the cluster's name (not the radio button).

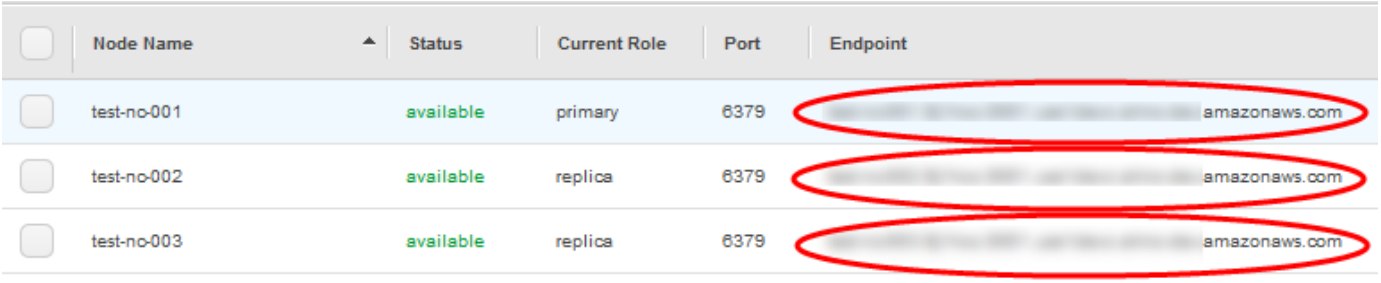



*Primary and Reader endpoints for a Valkey (cluster mode disabled) cluster*

If there is only one node in the cluster, there is no primary endpoint and you can continue at the next step.

4. If the Valkey (cluster mode disabled) cluster has replica nodes, you can find the cluster's replica node endpoints by choosing the cluster's name and then choosing the **Nodes** tab.

   The nodes screen appears with each node in the cluster, primary and replicas, listed with its endpoint.




*Node endpoints for a Valkey (cluster mode disabled) cluster*

5. To copy an endpoint to your clipboard:

   a. One endpoint at a time, find the endpoint you want to copy.

   b. Choose the copy icon directly in front of the endpoint.

   The endpoint is now copied to your clipboard. For information on using the endpoint to connect to a node, see Connecting to nodes.

A Valkey (cluster mode disabled) primary endpoint looks something like the following. There is a difference depending upon whether or not In-Transit encryption is enabled.

**In-transit encryption not enabled**

```
clusterName.xxxxxx.nodeId.regionAndAz.cache.amazonaws.com:port

redis-01.7abc2d.0001.usw2.cache.amazonaws.com:6379
```

**In-transit encryption enabled**

```
master.clusterName.xxxxxx.regionAndAz.cache.amazonaws.com:port

master.ncit.ameaqx.use1.cache.amazonaws.com:6379
```

To further explore how to find your endpoints, see the relevant topics for the engine and cluster type you're running.

- Finding connection endpoints in ElastiCache
- Finding Endpoints for a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (Console)—You need the cluster's Configuration endpoint.

- Finding Endpoints (AWS CLI)
- Finding Endpoints (ElastiCache API)

### Connect to a Valkey or Redis OSS cluster or replication group (Linux)

Now that you have the endpoint you need, you can log in to an EC2 instance and connect to the cluster or replication group. In the following example, you use the *valkey-cli* utility to connect to a cluster. The latest version of valkey-cli also supports SSL/TLS for connecting encryption/authentication enabled clusters.

The following example uses Amazon EC2 instances running Amazon Linux and Amazon Linux 2. For details on installing and compiling valkey-cli with other Linux distributions, see the documentation for your specific operating system..

> **Note**
>
> This process covers testing a connection using valkey-cli utility for unplanned use only. For a list of supported Valkey and Redis OSS clients, see the Valkey documentation. For examples of using the AWS SDKs with ElastiCache, see Tutorials: Getting started with Python and ElastiCache.

### Connecting to a cluster mode disabled unencrypted-cluster

1. Run the following command to connect to the cluster and replace *primary-endpoint* and *port number* with the endpoint of your cluster and your port number. (The default port for Valkey or Redis OSS is 6379.)

   ```
   src/valkey-cli -h primary-endpoint -p port number
   ```

   The result in a Valkey or Redis OSS command prompt looks similar to the following:

   ```
   primary-endpoint:port number
   ```

2. You can now run Valkey or Redis OSS commands.

   ```
   set x Hello
   OK
   ```

```
get x
"Hello"
```

### Connecting to a cluster mode enabled unencrypted-cluster

1. Run the following command to connect to the cluster and replace *configuration-endpoint* and *port number* with the endpoint of your cluster and your port number. (The default port for Valkey or Redis OSS is 6379.)

   ```
   src/valkey-cli -h configuration-endpoint -c -p port number
   ```

   > **Note**
   >
   > In the preceding command, option -c enables cluster mode following -ASK and -MOVED redirections.

   The result in a Valkey or Redis OSS command prompt looks similar to the following:

   ```
   configuration-endpoint:port number
   ```

2. You can now run Valkey or Redis OSS commands. Note that redirection occurs because you enabled it using the -c option. If redirection isn't enabled, the command returns the MOVED error. For more information on the MOVED error, see Redis OSS cluster specification.

   ```
   set x Hi
   -> Redirected to slot [16287] located at 172.31.28.122:6379
   OK
   set y Hello
   OK
   get y
   "Hello"
   set z Bye
   -> Redirected to slot [8157] located at 172.31.9.201:6379
   OK
   get z
   "Bye"
   get x
   -> Redirected to slot [16287] located at 172.31.28.122:6379
   ```

```
"Hi"
```

### Connecting to an Encryption/Authentication enabled cluster

By default, valkey-cli uses an unencrypted TCP connection when connecting to Valkey or Redis OSS. The option `BUILD_TLS=yes` enables SSL/TLS at the time of valkey-cli compilation as shown in the preceding Download and set up command line access section. Enabling AUTH is optional. However, you must enable encryption in-transit in order to enable AUTH. For more details on ElastiCache encryption and authentication, see ElastiCache in-transit encryption (TLS).

> **Note**
>
> You can use the option `--tls` with valkey-cli to connect to both cluster mode enabled and disabled encrypted clusters. If a cluster has an AUTH token set, then you can use the option `-a` to provide an AUTH password.

In the following examples, be sure to replace *`cluster-endpoint`* and *`port number`* with the endpoint of your cluster and your port number. (The default port for Valkey or Redis OSS is 6379.)

#### Connect to cluster mode disabled encrypted clusters

The following example connects to an encryption and authentication enabled cluster:

```
src/valkey-cli -h cluster-endpoint --tls -a your-password -p port number
```

The following example connects to a cluster that has only encryption enabled:

```
src/valkey-cli -h cluster-endpoint --tls -p port number
```

#### Connect to cluster mode enabled encrypted clusters

The following example connects to an encryption and authentication enabled cluster:

```
src/valkey-cli -c -h cluster-endpoint --tls -a your-password -p port number
```

The following example connects to a cluster that has only encryption enabled:

```
src/valkey-cli -c -h cluster-endpoint --tls -p port number
```

After you connect to the cluster, you can run the Valkey or Redis OSS commands as shown in the preceding examples for unencrypted clusters.

**valkey-cli alternative**

If the cluster isn't cluster mode enabled and you need to make a connection to the cluster for a short test but without going through the valkey-cli compilation, you can use telnet or openssl. In the following example commands, be sure to replace *cluster-endpoint* and *port number* with the endpoint of your cluster and your port number. (The default port for Valkey or Redis OSS is 6379.)

The following example connects to an encryption and/or authentication enabled cluster mode disabled cluster:

```
openssl s_client -connect cluster-endpoint:port number
```

If the cluster has a password set, first connect to the cluster. After connecting, authenticate the cluster using the following command, then press the `Enter` key. In the following example, replace *your-password* with the password for your cluster.

```
Auth your-password
```

The following example connects to a cluster mode disabled cluster that doesn't have encryption or authentication enabled:

```
telnet cluster-endpoint port number
```

**Connect to a Valkey or Redis OSS cluster or replication group (Windows)**

In order to connect to the Valkey or Redis OSS cluster from an EC2 Windows instance using the Valkey CLI or Redis OSS CLI, you must download the *valkey-cli* package and use *valkey-cli.exe* to connect to the Valkey or Redis OSS cluster from an EC2 Windows instance.

In the following example, you use the *valkey-cli* utility to connect to a cluster that is not encryption enabled and running Valkey or Redis OSS. For more information about Valkey or Redis OSS and available commands, see Valkey and Redis OSS commands on the Valkey website.

**To connect to a Valkey or Redis OSS cluster that is not encryption-enabled using *valkey-cli***

1. Connect to your Amazon EC2 instance using the connection utility of your choice. For instructions on how to connect to an Amazon EC2 instance, see the Amazon EC2 Getting Started Guide.
2. Copy and paste the link https://github.com/microsoftarchive/redis/releases/download/win-3.0.504/Redis-x64-3.0.504.zip in an Internet browser to download the zip file for the Valkey client from the available release at GitHub https://github.com/microsoftarchive/redis/releases/tag/win-3.0.504

   Extract the zip file to you desired folder/path.

   Open the Command Prompt and change to the Valkey directory and run the command `c:\Valkey>valkey-cli -h` *`Redis_Cluster_Endpoint`* `-p 6379`.

   For example:

   ```
   c:\Valkey>valkey-cli -h cmd.xxxxxxx.ng.0001.usw2.cache.amazonaws.com -p 6379
   ```
3. Run Valkey or Redis OSS commands.

   You are now connected to the cluster and can run Valkey or Redis OSS commands like the following.

   ```
   set a "hello"          // Set key "a" with a string value and no expiration
   OK
   get a                  // Get value for key "a"
   "hello"
   get b                  // Get value for key "b" results in miss
   (nil)
   set b "Good-bye" EX 5  // Set key "b" with a string value and a 5 second expiration
   "Good-bye"
   get b                  // Get value for key "b"
   "Good-bye"
                          // wait >= 5 seconds
   get b
   (nil)                  // key has expired, nothing returned
   quit                   // Exit from valkey-cli
   ```

### Where do I go from here?

Now that you have tried the Getting Started exercise, you can explore the following sections to learn more about ElastiCache and available tools:

- Getting started with AWS
- Tools for Amazon Web Services
- AWS Command Line Interface
- Amazon ElastiCache API reference

After you complete the Getting Started exercise, you can read these sections to learn more about ElastiCache administration:

- Choosing your node size

  You want your cache to be large enough to accommodate all the data you want to cache. At the same time, you don't want to pay for more cache than you need. Use this topic to help you choose the best node size.

- ElastiCache best practices and caching strategies

  Identify and address issues that can affect the efficiency of your cluster.

## Creating a node-based ElastiCache cluster for Redis OSS

Following are the one-time actions you must take in order to create a node-based ElastiCache cluster for Redis OSS.

For further information on setting up ElastiCache see Setting up ElastiCache.

**Topics**

- Step 1: Create a subnet group
- Step 2: Create a cluster
- Step 3: Authorize access to the cluster
- Step 4: Connect to the cluster's node

## Step 1: Create a subnet group

Before you create a cluster, you first create a subnet group. A *cache subnet group* is a collection of subnets that you may want to designate for your clusters in a VPC. When launching a cluster in a VPC, you need to select a cache subnet group. Then ElastiCache uses that cache subnet group to assign IP addresses within that subnet to each cache node in the cluster.

When you create a new subnet group, note the number of available IP addresses. If the subnet has very few free IP addresses, you might be constrained as to how many more nodes you can add to the cluster. To resolve this issue, you can assign one or more subnets to a subnet group so that you have a sufficient number of IP addresses in your cluster's Availability Zone. After that, you can add more nodes to your cluster.

The following procedures show you how to create a subnet group called `mysubnetgroup` (console) and the AWS CLI.

### Creating a subnet group (Console)

The following procedure shows how to create a subnet group (console).

**To create a subnet group (Console)**

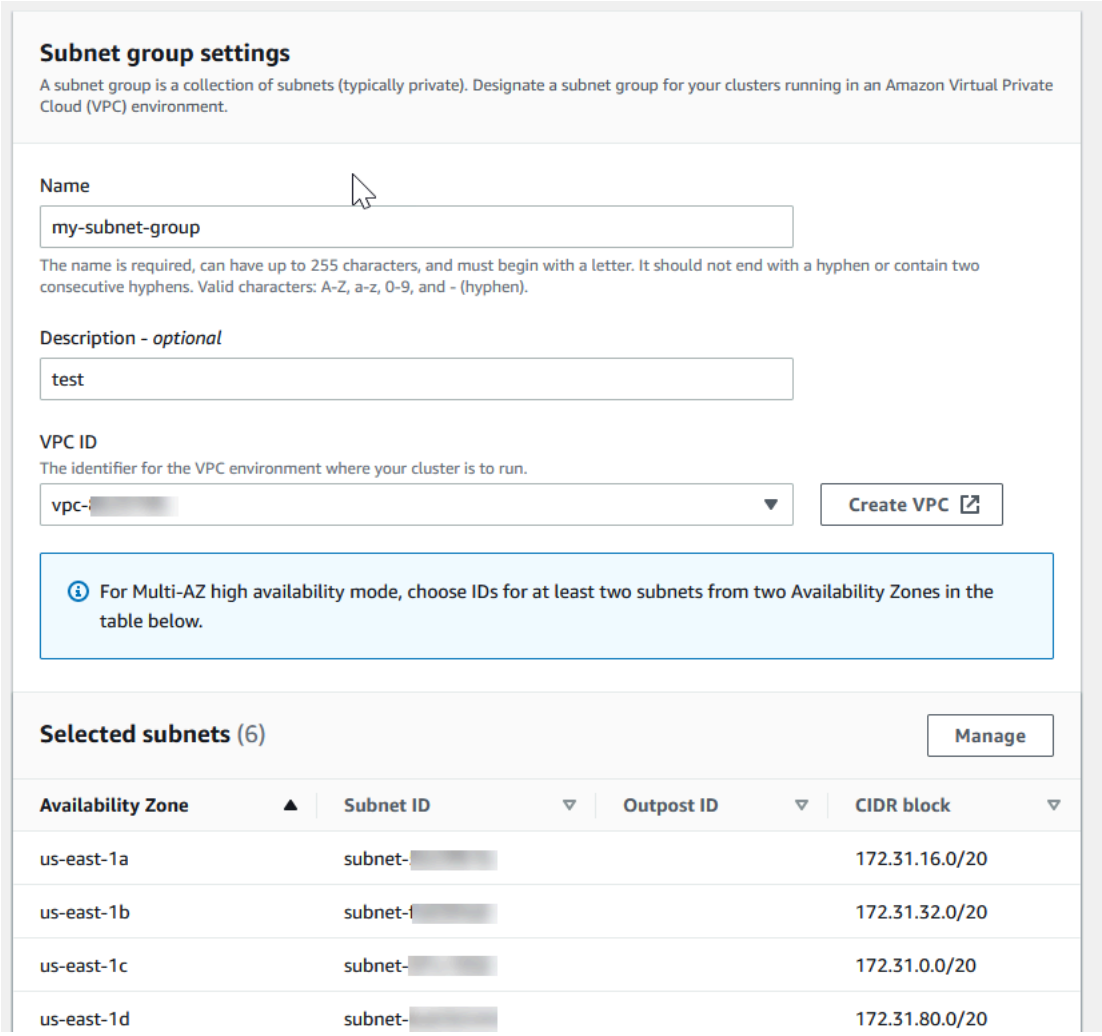
1. Sign in to the AWS Management Console, and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation list, choose **Subnet Groups**.
3. Choose **Create Subnet Group**.
4. In the **Create Subnet Group** wizard, do the following. When all the settings are as you want them, choose **Yes, Create**.

    a. In the **Name** box, type a name for your subnet group.

    b. In the **Description** box, type a description for your subnet group.

    c. In the **VPC ID** box, choose the Amazon VPC that you created.

    d. In the **Availability Zone** and **Subnet ID** lists, choose the Availability Zone or Local Zone and ID of your private subnet, and then choose **Add**.




5. In the confirmation message that appears, choose **Close**.

Your new subnet group appears in the **Subnet Groups** list of the ElastiCache console. At the bottom of the window you can choose the subnet group to see details, such as all of the subnets associated with this group.

### Create a subnet group (AWS CLI)

At a command prompt, use the command `create-cache-subnet-group` to create a subnet group.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-subnet-group \
    --cache-subnet-group-name mysubnetgroup \
    --cache-subnet-group-description "Testing" \
```

```
    --subnet-ids subnet-53df9c3a
```

For Windows:

```
aws elasticache create-cache-subnet-group ^
    --cache-subnet-group-name mysubnetgroup ^
    --cache-subnet-group-description "Testing" ^
    --subnet-ids subnet-53df9c3a
```

This command should produce output similar to the following:

```
{
    "CacheSubnetGroup": {
        "VpcId": "vpc-37c3cd17",
        "CacheSubnetGroupDescription": "Testing",
        "Subnets": [
            {
                "SubnetIdentifier": "subnet-53df9c3a",
                "SubnetAvailabilityZone": {
                    "Name": "us-west-2a"
                }
            }
        ],
        "CacheSubnetGroupName": "mysubnetgroup"
    }
}
```

For more information, see the AWS CLI topic create-cache-subnet-group.

## Step 2: Create a cluster

Before creating a cluster for production use, you obviously need to consider how you will configure the cluster to meet your business needs. Those issues are addressed in the Preparing a cluster in ElastiCache section. For the purposes of this Getting Started exercise, you will create a cluster with cluster mode disabled and you can accept the default configuration values where they apply.

The cluster you create will be live, and not running in a sandbox. You will incur the standard ElastiCache usage fees for the instance until you delete it. The total charges will be minimal (typically less than a dollar) if you complete the exercise described here in one sitting and delete your cluster when you are finished. For more information about ElastiCache usage rates, see Amazon ElastiCache.

Your cluster is launched in a virtual private cloud (VPC) based on the Amazon VPC service.

### Creating a Redis OSS (cluster mode disabled) cluster (Console)

#### To create a Redis OSS (cluster mode disabled) cluster using the ElastiCache console

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region that you want to launch this cluster in.
3. Choose **Get started** from the navigation pane.
4. Choose **Create VPC** and follow the steps outlined at Creating a Virtual Private Cloud (VPC).
5. On the ElastiCache dashboard page, choose **Valkey cache** or **Redis OSS cache**. For this exercise we will choose **Redis OSS cache**, and then choose **Create Redis OSS cache**.
6. Under **Cluster settings**, do the following:

   a. Choose **Configure and create a new cluster**.

   b. For **Cluster mode**, choose **Disabled**.

   c. For **Cluster info** enter a value for **Name**.

   d. (Optional) Enter a value for **Description**.
7. Under **Location**:

   AWS Cloud

   1. For **AWS Cloud**, we recommend you accept the default settings for **Multi-AZ** and **Auto-failover**. For more information, see Minimizing downtime in ElastiCache for Redis OSS with Multi-AZ.
   2. Under **Cluster settings**

      a. For **Engine version**, choose an available version.

      b. For **Port**, use the default port, 6379. If you have a reason to use a different port, enter the port number.

      c. For **Parameter group**, choose a parameter group or create a new one. Parameter groups control the runtime parameters of your cluster. For more information on parameter groups, see Valkey and Redis OSS parameters and Creating an ElastiCache parameter group.

> **Note**
>
> When you select a parameter group to set the engine configuration values, that parameter group is applied to all clusters in the global datastore. On the **Parameter Groups** page, the yes/no **Global** attribute indicates whether a parameter group is part of a global datastore.

d. For **Node type**, choose the down arrow (▼ ).
In the **Change node type** dialog box, choose a value for **Instance family** for the node type that you want. Then choose the node type that you want to use for this cluster, and then choose **Save**.

For more information, see Choosing your node size.

If you choose a data tiering node type, ElastiCache automatically enables data tiering. For more information, see Data tiering in ElastiCache.

e. For **Number of replicas**, choose the number of read replicas you want. If you enabled Multi-AZ, the number must be between 1-5.

3. Under **Connectivity**

a. For **Network type**, choose the IP version(s) this cluster will support.

b. For **Subnet groups**, choose the subnet that you want to apply to this cluster. ElastiCache uses that subnet group to choose a subnet and IP addresses within that subnet to associate with your nodes. ElastiCache clusters require a dual-stack subnet with both IPv4 and IPv6 addresses assigned to them to operate in dual-stack mode and an IPv6-only subnet to operate as IPv6-only.

When creating a new subnet group, enter the **VPC ID** to which it belongs.

For more information, see:

- Choosing a network type in ElastiCache.
- Create a subnet in your VPC.

If you are Using local zones with ElastiCache, you must create or choose a subnet that is in the local zone.

For more information, see Subnets and subnet groups.

4. For **Availability zone placements**, you have two options:

   - **No preference** – ElastiCache chooses the Availability Zone.
   - **Specify availability zones** – You specify the Availability Zone for each cluster.

     If you chose to specify the Availability Zones, for each cluster in each shard, choose the Availability Zone from the list.

   For more information, see Choosing regions and availability zones for ElastiCache.

5. Choose **Next**

6. Under **Advanced Redis OSS settings**

   - For **Security**:

     i. To encrypt your data, you have the following options:

     - **Encryption at rest** – Enables encryption of data stored on disk. For more information, see Encryption at Rest.

       > **Note**
       >
       > You have the option to supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key. For more information, see Using customer managed keys from AWS KMS.

     - **Encryption in-transit** – Enables encryption of data on the wire. For more information, see encryption in transit. For ElastiCache engine version 6.0 for Redis OSS and above, if you enable encryption in-transit you will be prompted to specify one of the following **Access Control** options:
       - **No Access Control** – This is the default setting. This indicates no restrictions on user access to the cluster.
       - **User Group Access Control List** – Select a user group with a defined set of users that can access the cluster. For more information, see Managing User Groups with the Console and CLI.

- **AUTH Default User** – An authentication mechanism for Valkey and Redis OSS server. For more information, see AUTH.
- **AUTH** – An authentication mechanism for Redis OSS server. For more information, see AUTH.

> **Note**
>
> RBAC is available on Valkey, and on Redis OSS version 6.0 and later. On earlier Redis OSS versions, AUTH is the sole option.

ii. For **Security groups**, choose the security groups that you want for this cluster. A *security group* acts as a firewall to control network access to your cluster. You can use the default security group for your VPC or create a new one.

For more information on security groups, see Security groups for your VPC in the *Amazon VPC User Guide.*

7. For regularly scheduled automatic backups, select **Enable automatic backups** and then enter the number of days that you want each automatic backup retained before it is automatically deleted. If you don't want regularly scheduled automatic backups, clear the **Enable automatic backups** check box. In either case, you always have the option to create manual backups.

   For more information on backup and restore, see Snapshot and restore.

8. (Optional) Specify a maintenance window. The *maintenance window* is the time, generally an hour in length, each week when ElastiCache schedules system maintenance for your cluster. You can allow ElastiCache to choose the day and time for your maintenance window (*No preference*), or you can choose the day, time, and duration yourself (*Specify maintenance window*). If you choose *Specify maintenance window* from the lists, choose the *Start day*, *Start time*, and *Duration* (in hours) for your maintenance window. All times are UCT times.

   For more information, see Managing ElastiCache cluster maintenance.

9. (Optional) For **Logs**:

   - Under **Log format**, choose either **Text** or **JSON**.
   - Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.

- Under **Log destination**, choose either **Create new** and enter either your CloudWatch Logs log group name or your Firehose stream name, or choose **Select existing** and then choose either your CloudWatch Logs log group name or your Firehose stream name,

10. For **Tags**, to help you manage your clusters and other ElastiCache resources, you can assign your own metadata to each resource in the form of tags. For mor information, see Tagging your ElastiCache resources.
11. Choose **Next**.
12. Review all your entries and choices, then make any needed corrections. When you're ready, choose **Create**.

On premises

1. For **On premises**, we recommend you leave **Auto-failover** enabled. For more information, see Minimizing downtime in ElastiCache for Redis OSS with Multi-AZ
2. To finish creating the cluster, follow the steps at Using Outposts.

As soon as your cluster's status is *available*, you can grant Amazon EC2 access to it, connect to it, and begin using it. For more information, see Step 3. Authorize access to the cluster and Step 4. Connect to the cluster's node.

> **Important**
>
> Once your cluster is available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Creating a Redis OSS (cluster mode disabled) cluster (AWS CLI)

#### Example

The following CLI code creates a Redis OSS (cluster mode disabled) cluster with no replicas.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
```

```
--cache-cluster-id my-cluster \
--cache-node-type cache.r4.large \
--engine redis \
--num-cache-nodes 1 \
--snapshot-arns arn:aws:s3:::my_bucket/snapshot.rdb
```

For Windows:

```
aws elasticache create-cache-cluster ^
--cache-cluster-id my-cluster ^
--cache-node-type cache.r4.large ^
--engine redis ^
--num-cache-nodes 1 ^
--snapshot-arns arn:aws:s3:::my_bucket/snapshot.rdb
```

To work with cluster mode enabled, see the following topics:

- To use the console, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console).
- To use the AWS CLI, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (AWS CLI).

## Step 3: Authorize access to the cluster

This section assumes that you are familiar with launching and connecting to Amazon EC2 instances. For more information, see the *Amazon EC2 Getting Started Guide*.

All ElastiCache clusters are designed to be accessed from an Amazon EC2 instance. The most common scenario is to access an ElastiCache cluster from an Amazon EC2 instance in the same Amazon Virtual Private Cloud (Amazon VPC), which will be the case for this exercise.

By default, network access to your cluster is limited to the account that was used to create it. Before you can connect to a cluster from an EC2 instance, you must authorize the EC2 instance to access the cluster. The steps required depend upon whether you launched your cluster into EC2-VPC or EC2-Classic.

The most common use case is when an application deployed on an EC2 instance needs to connect to a cluster in the same VPC. The simplest way to manage access between EC2 instances and clusters in the same VPC is to do the following:

1. Create a VPC security group for your cluster. This security group can be used to restrict access to the cluster instances. For example, you can create a custom rule for this security group that allows TCP access using the port you assigned to the cluster when you created it and an IP address you will use to access the cluster.

   The default port for Redis OSS clusters and replication groups is 6379.

   > **Important**
   >
   > Amazon ElastiCache security groups are only applicable to clusters that are *not* running in an Amazon Virtual Private Cloud environment (VPC). If you are running in an Amazon Virtual Private Cloud, **Security Groups** is not available in the console navigation pane.
   > If you are running your ElastiCache nodes in an Amazon VPC, you control access to your clusters with Amazon VPC security groups, which are different from ElastiCache security groups. For more information about using ElastiCache in an Amazon VPC, see Amazon VPCs and ElastiCache security

2. Create a VPC security group for your EC2 instances (web and application servers). This security group can, if needed, allow access to the EC2 instance from the Internet via the VPC's routing

table. For example, you can set rules on this security group to allow TCP access to the EC2 instance over port 22.

3. Create custom rules in the security group for your Cluster that allow connections from the security group you created for your EC2 instances. This would allow any member of the security group to access the clusters.

> **Note**
>
> If you are planning to use Using local zones with ElastiCache, ensure that you have enabled them. When you create a subnet group in that local zone, your VPC is extended to that Local Zone and your VPC will treat the subnet as any subnet in any other Availability Zone. All relevant gateways and route tables will be automatically adjusted.

**To create a rule in a VPC security group that allows connections from another security group**

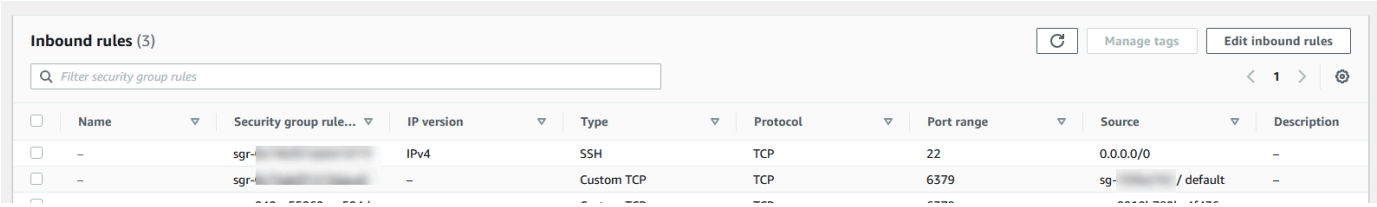
1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc.
2. In the navigation pane, choose **Security Groups**.
3. Select or create a security group that you will use for your Cluster instances. Under **Inbound Rules**, select **Edit Inbound Rules** and then select **Add Rule**. This security group will allow access to members of another security group.
4. From **Type** choose **Custom TCP Rule**.

   a. For **Port Range**, specify the port you used when you created your cluster.

      The default port for Redis OSS clusters and replication groups is 6379.

   b. In the **Source** box, start typing the ID of the security group. From the list select the security group you will use for your Amazon EC2 instances.

5. Choose **Save** when you finish.



Once you have enabled access, you are now ready to connect to the node, as discussed in the next section.

For information on accessing your ElastiCache cluster from a different Amazon VPC, a different AWS Region, or even your corporate network, see the following:

- Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC
- Accessing ElastiCache resources from outside AWS

## Step 4: Connect to the cluster's node

Before you continue, complete Step 3: Authorize access to the cluster.

This section assumes that you've created an Amazon EC2 instance and can connect to it. For instructions on how to do this, see the Amazon EC2 Getting Started Guide.

An Amazon EC2 instance can connect to a cluster node only if you have authorized it to do so.

### Find your node endpoints

When your cluster is in the *available* state and you've authorized access to it, you can log in to an Amazon EC2 instance and connect to the cluster. To do so, you must first determine the endpoint.

### Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)

If a Redis OSS (cluster mode disabled) cluster has only one node, the node's endpoint is used for both reads and writes. If the cluster has multiple nodes, there are three types of endpoints; the *primary endpoint*, the *reader endpoint* and the *node endpoints*.

The primary endpoint is a DNS name that always resolves to the primary node in the cluster. The primary endpoint is immune to changes to your cluster, such as promoting a read replica to the primary role. For write activity, we recommend that your applications connect to the primary endpoint.

A reader endpoint will evenly split incoming connections to the endpoint between all read replicas in a ElastiCache for Redis OSS cluster. Additional factors such as when the application creates the connections or how the application (re)-uses the connections will determine the traffic distribution. Reader endpoints keep up with cluster changes in real-time as replicas are added or removed. You can place your ElastiCache for Redis OSS cluster's multiple read replicas in different AWS Availability Zones (AZ) to ensure high availability of reader endpoints.

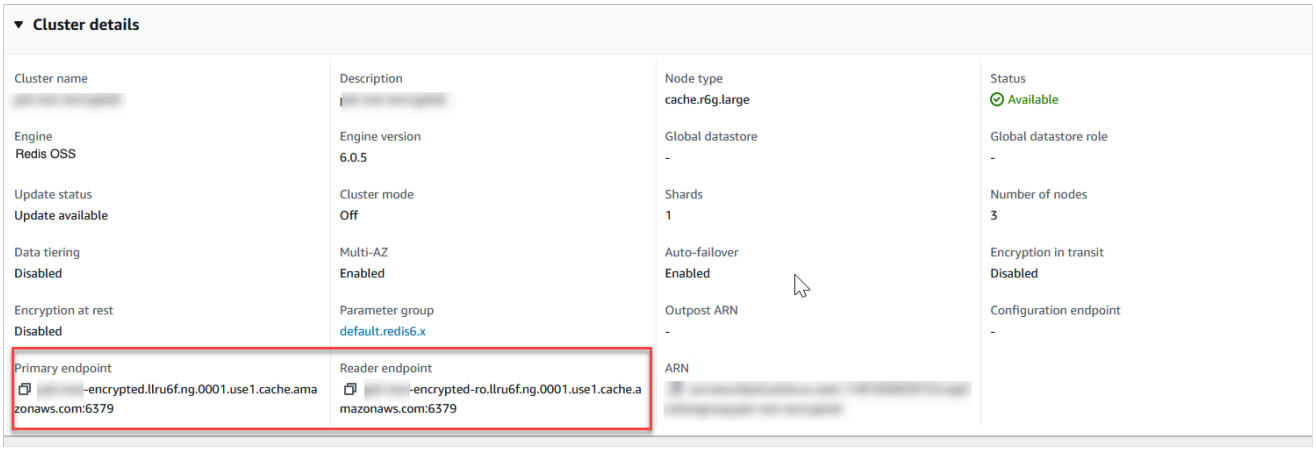
> **Note**
>
> A reader endpoint is not a load balancer. It is a DNS record that will resolve to an IP address of one of the replica nodes in a round robin fashion.

For read activity, applications can also connect to any node in the cluster. Unlike the primary endpoint, node endpoints resolve to specific endpoints. If you make a change in your cluster, such as adding or deleting a replica, you must update the node endpoints in your application.

**To find a Redis OSS (cluster mode disabled) cluster's endpoints**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Redis OSS caches**.

   The clusters screen will appear with a list that will include any existing Valkey or Redis OSS serverless caches, Redis OSS (cluster mode disabled) clusters and Redis OSS (cluster mode enabled) clusters. Choose the cluster you created in the Creating a Redis OSS (cluster mode disabled) cluster (Console) section.
3. To find the cluster's Primary and/or Reader endpoints, choose the cluster's name (not the radio button).

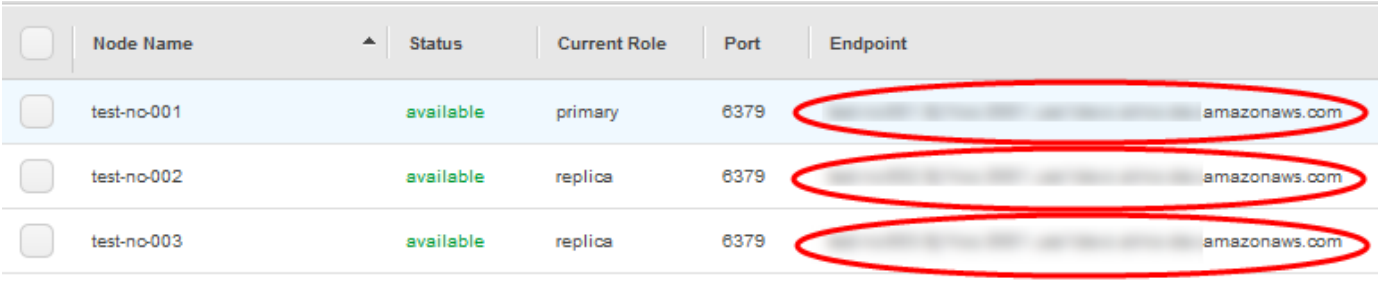



*Primary and Reader endpoints for a Redis OSS (cluster mode disabled) cluster*

If there is only one node in the cluster, there is no primary endpoint and you can continue at the next step.

4. If the Redis OSS (cluster mode disabled) cluster has replica nodes, you can find the cluster's replica node endpoints by choosing the cluster's name and then choosing the **Nodes** tab.

   The nodes screen appears with each node in the cluster, primary and replicas, listed with its endpoint.




*Node endpoints for a Redis OSS (cluster mode disabled) cluster*

5. To copy an endpoint to your clipboard:

   a. One endpoint at a time, find the endpoint you want to copy.

   b. Choose the copy icon directly in front of the endpoint.

   The endpoint is now copied to your clipboard. For information on using the endpoint to connect to a node, see Connecting to nodes.

A Redis OSS (cluster mode disabled) primary endpoint looks something like the following. There is a difference depending upon whether or not In-Transit encryption is enabled.

**In-transit encryption not enabled**

```
clusterName.xxxxxx.nodeId.regionAndAz.cache.amazonaws.com:port

redis-01.7abc2d.0001.usw2.cache.amazonaws.com:6379
```

**In-transit encryption enabled**

```
master.clusterName.xxxxxx.regionAndAz.cache.amazonaws.com:port

master.ncit.ameaqx.use1.cache.amazonaws.com:6379
```

To further explore how to find your endpoints, see the relevant topics for the engine and cluster type you're running.

- Finding connection endpoints in ElastiCache
- Finding Endpoints for a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (Console)—You need the cluster's Configuration endpoint.

- Finding Endpoints (AWS CLI)
- Finding Endpoints (ElastiCache API)

### Connect to a Valkey or Redis OSS cluster or replication group (Linux)

Now that you have the endpoint you need, you can log in to an EC2 instance and connect to the cluster or replication group. In the following example, you use the *valkey-cli* utility to connect to a cluster. The latest version of valkey-cli also supports SSL/TLS for connecting encryption/authentication enabled clusters.

The following example uses Amazon EC2 instances running Amazon Linux and Amazon Linux 2. For details on installing and compiling valkey-cli with other Linux distributions, see the documentation for your specific operating system..

> **Note**
>
> This process covers testing a connection using valkey-cli utility for unplanned use only. For a list of supported clients, see the Valkey documentation. For examples of using the AWS SDKs with ElastiCache, see Tutorials: Getting started with Python and ElastiCache.

### Connecting to a cluster mode disabled unencrypted-cluster

1. Run the following command to connect to the cluster and replace *primary-endpoint* and *port number* with the endpoint of your cluster and your port number. (The default port for Valkey and Redis OSS is 6379.)

   ```
   src/valkey-cli -h primary-endpoint -p port number
   ```

   The result in a command prompt looks similar to the following:

   ```
   primary-endpoint:port number
   ```

2. You can now run Valkey and Redis OSS commands.

   ```
   set x Hello
   OK

   get x
   ```

```
"Hello"
```

### Connecting to a cluster mode enabled unencrypted-cluster

1. Run the following command to connect to the cluster and replace *configuration-endpoint* and *port number* with the endpoint of your cluster and your port number. (The default port for Valkey and Redis OSS is 6379.)

   ```
   src/valkey-cli -h configuration-endpoint -c -p port number
   ```

   > **Note**
   >
   > In the preceding command, option -c enables cluster mode following -ASK and -MOVED redirections.

   The result in a command prompt looks similar to the following:

   ```
   configuration-endpoint:port number
   ```

2. You can now run Valkey and Redis OSS commands. Note that redirection occurs because you enabled it using the -c option. If redirection isn't enabled, the command returns the MOVED error. For more information on the MOVED error, see cluster specification.

   ```
   set x Hi
   -> Redirected to slot [16287] located at 172.31.28.122:6379
   OK
   set y Hello
   OK
   get y
   "Hello"
   set z Bye
   -> Redirected to slot [8157] located at 172.31.9.201:6379
   OK
   get z
   "Bye"
   get x
   -> Redirected to slot [16287] located at 172.31.28.122:6379
   "Hi"
   ```

### Connecting to an Encryption/Authentication enabled cluster

By default, valkey-cli uses an unencrypted TCP connection when connecting to Valkey and Redis OSS. The option `BUILD_TLS=yes` enables SSL/TLS at the time of valkey-cli compilation as shown in the preceding Download and set up command line access section. Enabling AUTH is optional. However, you must enable encryption in-transit in order to enable AUTH. For more details on ElastiCache encryption and authentication, see ElastiCache in-transit encryption (TLS).

> **Note**
>
> You can use the option `--tls` with valkey-cli to connect to both cluster mode enabled and disabled encrypted clusters. If a cluster has an AUTH token set, then you can use the option `-a` to provide an AUTH password.

In the following examples, be sure to replace *`cluster-endpoint`* and *`port number`* with the endpoint of your cluster and your port number. (The default port for Redis OSS is 6379.)

#### Connect to cluster mode disabled encrypted clusters

The following example connects to an encryption and authentication enabled cluster:

```
src/valkey-cli -h cluster-endpoint --tls -a your-password -p port number
```

The following example connects to a cluster that has only encryption enabled:

```
src/valkey-cli -h cluster-endpoint --tls -p port number
```

#### Connect to cluster mode enabled encrypted clusters

The following example connects to an encryption and authentication enabled cluster:

```
src/valkey-cli -c -h cluster-endpoint --tls -a your-password -p port number
```

The following example connects to a cluster that has only encryption enabled:

```
src/valkey-cli -c -h cluster-endpoint --tls -p port number
```

After you connect to the cluster, you can run the Valkey or Redis OSS commands as shown in the preceding examples for unencrypted clusters.

### An alternative to valkey-cli or Redis-cli

If the cluster isn't cluster mode enabled and you need to make a connection to the cluster for a short test but without going through the valkey-cli or redis-cli compilation, you can use telnet or openssl. In the following example commands, be sure to replace *cluster-endpoint* and *port number* with the endpoint of your cluster and your port number. (The default port for Redis OSS is 6379.)

The following example connects to an encryption and/or authentication enabled cluster mode disabled cluster:

```
openssl s_client -connect cluster-endpoint:port number
```

If the cluster has a password set, first connect to the cluster. After connecting, authenticate the cluster using the following command, then press the `Enter` key. In the following example, replace *your-password* with the password for your cluster.

```
Auth your-password
```

The following example connects to a cluster mode disabled cluster that doesn't have encryption or authentication enabled:

```
telnet cluster-endpoint port number
```

### Connect to a Valkey or Redis OSS cluster or replication group (Windows)

In order to connect to the cluster from an EC2 Windows instance using the Valkey or Redis OSS CLI, you must download the *valkey-cli* package and use *valkey-cli.exe* to connect to the Valkey or Redis OSS cluster from an EC2 Windows instance.

In the following example, you use the *valkey-cli* utility to connect to a cluster that is not encryption enabled and running Valkey or Redis OSS. For more information about Valkey and available commands, see Valkey commands on the Valkey website.

#### To connect to a Valkey or Redis OSS cluster that is not encryption-enabled using *valkey-cli*

1. Connect to your Amazon EC2 instance using the connection utility of your choice. For instructions on how to connect to an Amazon EC2 instance, see the Amazon EC2 Getting Started Guide.

2. Copy and paste the link https://github.com/microsoftarchive/redis/releases/download/win-3.0.504/Redis-x64-3.0.504.zip in an Internet browser to download the zip file for the Redis OSS client from the available release at GitHub https://github.com/microsoftarchive/redis/releases/tag/win-3.0.504

   Extract the zip file to you desired folder/path.

   Open the Command Prompt and change to the Valkey directory and run the command `c:\Valkey>valkey-cli -h Valkey_Cluster_Endpoint -p 6379`.

   For example:

   ```
   c:\Valkey>valkey-cli -h cmd.xxxxxxx.ng.0001.usw2.cache.amazonaws.com -p 6379
   ```

3. Run Valkey or Redis OSS commands.

   You are now connected to the cluster and can run Valkey or Redis OSS commands like the following.

   ```
   set a "hello"          // Set key "a" with a string value and no expiration
   OK
   get a                  // Get value for key "a"
   "hello"
   get b                  // Get value for key "b" results in miss
   (nil)
   set b "Good-bye" EX 5  // Set key "b" with a string value and a 5 second expiration
   "Good-bye"
   get b                  // Get value for key "b"
   "Good-bye"
                          // wait >= 5 seconds
   get b
   (nil)                  // key has expired, nothing returned
   quit                   // Exit from valkey-cli
   ```

# Deleting a cluster

As long as a cluster is in the *available* state, you are being charged for it, whether or not you are actively using it. To stop incurring charges, delete the cluster.

**Warning**

- When you delete an ElastiCache cluster, your manual snapshots are retained. You can also create a final snapshot before the cluster is deleted. Automatic cache snapshots are not retained. For more information, see Snapshot and restore.
- `CreateSnapshot` permission is required to create a final snapshot. Without this permission, the API call will fail with an `Access Denied` exception.

## Using the AWS Management Console

The following procedure deletes a single cluster from your deployment. To delete multiple clusters, repeat the procedure for each cluster that you want to delete. You do not need to wait for one cluster to finish deleting before starting the procedure to delete another cluster.

**To delete a cluster**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the ElastiCache engine dashboard, choose Valkey or Redis OSS.

   A list of all caches running on that engine appears.
3. To choose the cluster to delete, choose the cluster's name from the list of clusters. In this case, the name of the cluster you created at Step 2: Create a cluster.

   **Important**

   You can only delete one cluster at a time from the ElastiCache console. Choosing multiple clusters disables the delete operation.
4. For **Actions**, choose **Delete**.
5. In the **Delete Cluster** confirmation screen, type the name of the cluster and choose **Final Backup**. Then choose **Delete** to delete the cluster, or choose **Cancel** to keep the cluster.

   If you chose **Delete**, the status of the cluster changes to *deleting*.

As soon as your cluster is no longer listed in the list of clusters, you stop incurring charges for it.

### Using the AWS CLI

The following code deletes the cluster `my-cluster`. In this case, substitute `my-cluster` with the name of the cluster you created at Step 2: Create a cluster.

```
aws elasticache delete-cache-cluster --cache-cluster-id my-cluster
```

The `delete-cache-cluster` CLI action only deletes one cluster. To delete multiple clusters, call `delete-cache-cluster` for each cache cluster that you want to delete. You do not need to wait for one cluster to finish deleting before deleting another.

For Linux, macOS, or Unix:

```
aws elasticache delete-cache-cluster \
    --cache-cluster-id my-cluster \
    --region us-east-2
```

For Windows:

```
aws elasticache delete-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --region us-east-2
```

For more information, see the AWS CLI for ElastiCache topic `delete-cache-cluster`.

## Other ElastiCache tutorials and videos

The following tutorials address tasks of interest to the Amazon ElastiCache user.

- ElastiCache Videos
- Tutorial: Configuring a Lambda Function to Access Amazon ElastiCache in an Amazon VPC

# ElastiCache Videos

Here are videos to help you learn basic and advanced Amazon ElastiCache concepts. For information about AWS Training, see AWS Training & Certification.

**Topics**


- Introductory Videos
- Advanced Videos


## Introductory Videos

The following videos introduce you to Amazon ElastiCache.

**Topics**


- AWS re:Invent 2020: What's new in Amazon ElastiCache
- AWS re:Invent 2019: What's new in Amazon ElastiCache
- AWS re:Invent 2017: What's new in Amazon ElastiCache
- DAT204—Building Scalable Applications on AWS NoSQL Services (re:Invent 2015)
- DAT207—Accelerating Application Performance with Amazon ElastiCache (AWS re:Invent 2013)


### AWS re:Invent 2020: What's new in Amazon ElastiCache

AWS re:Invent 2020: What's new in Amazon ElastiCache

### AWS re:Invent 2019: What's new in Amazon ElastiCache

AWS re:Invent 2019: What's new in Amazon ElastiCache

### AWS re:Invent 2017: What's new in Amazon ElastiCache

AWS re:Invent 2017: What's new in Amazon ElastiCache

### DAT204—Building Scalable Applications on AWS NoSQL Services (re:Invent 2015)

In this session, we discuss the benefits of NoSQL databases and take a tour of the main NoSQL services offered by AWS—Amazon DynamoDB and Amazon ElastiCache. Then, we hear from two

leading customers, Expedia and Mapbox, about their use cases and architectural challenges, and how they addressed them using AWS NoSQL services, including design patterns and best practices. You should come out of this session having a better understanding of NoSQL and its powerful capabilities, ready to tackle your database challenges with confidence.

DAT204—Building Scalable Applications on AWS NoSQL Services (re:Invent 2015)

**DAT207—Accelerating Application Performance with Amazon ElastiCache (AWS re:Invent 2013)**

In this video, learn how you can use Amazon ElastiCache to easily deploy an in-memory caching system to speed up your application performance. We show you how to use Amazon ElastiCache to improve your application latency and reduce the load on your database servers. We'll also show you how to build a caching layer that is easy to manage and scale as your application grows. During this session, we go over various scenarios and use cases that can benefit by enabling caching, and discuss the features provided by Amazon ElastiCache.

DAT207 - Accelerating Application Performance with Amazon ElastiCache (re:Invent 2013)

## Advanced Videos

The following videos cover more advanced Amazon ElastiCache topics.

**Topics**

- Design for success with Amazon ElastiCache best practices (re:Invent 2020)
- Supercharge your real-time apps with Amazon ElastiCache (re:Invent 2019)
- Best practices: migrating Redis OSS clusters from Amazon EC2 to ElastiCache (re:Invent 2019)
- Scaling a Fantasy Sports Platform with Amazon ElastiCache & Amazon Aurora STP11 (re:Invent 2018)
- Reliable & Scalable Redis OSS in the Cloud with Amazon ElastiCache (re:Invent 2018)
- ElastiCache Deep Dive: Design Patterns for In-Memory Data Stores (re:Invent 2018)
- DAT305—Amazon ElastiCache Deep Dive (re:Invent 2017)
- DAT306—Amazon ElastiCache Deep Dive (re:Invent 2016)
- DAT317—How IFTTT uses ElastiCache for Redis OSS to Predict Events (re:Invent 2016)
- DAT407—Amazon ElastiCache Deep Dive (re:Invent 2015)
- SDD402—Amazon ElastiCache Deep Dive (re:Invent 2014)

- DAT307—Deep Dive into Amazon ElastiCache Architecture and Design Patterns (re:Invent 2013)

**Design for success with Amazon ElastiCache best practices (re:Invent 2020)**

With the explosive growth of business-critical, real-time applications built on Redis OSS, availability, scalability, and security have become top considerations. Learn best practices for setting up Amazon ElastiCache for success with online scaling, high availability across Multi-AZ deployments, and security configurations.

Design for success with Amazon ElastiCache best practices (re:Invent 2020)

**Supercharge your real-time apps with Amazon ElastiCache (re:Invent 2019)**

With the rapid growth in cloud adoption and the new scenarios that it empowers, applications need microsecond latency and high throughput to support millions of requests per second. Developers have traditionally relied on specialized hardware and workarounds, such as disk-based databases combined with data reduction techniques, to manage data for real-time applications. These approaches can be expensive and not scalable. Learn how you can boost the performance of real-time applications by using the fully managed, in-memory Amazon ElastiCache for extreme performance, high scalability, availability, and security.

Supercharge your real-time apps with Amazon ElastiCache (re:Invent 2019:)

**Best practices: migrating Redis OSS clusters from Amazon EC2 to ElastiCache (re:Invent 2019)**

Managing Redis OSS clusters on your own can be hard. You have to provision hardware, patch software, back up data, and monitor workloads constantly. With the newly released Online Migration feature for Amazon ElastiCache, you can now easily move your data from self-hosted Redis OSS on Amazon EC2 to fully managed Amazon ElastiCache, with cluster mode disabled. In this session, you learn about the new Online Migration tool, see a demo, and, more importantly, learn hands-on best practices for a smooth migration to Amazon ElastiCache.

Best practices: migrating Redis OSS clusters from Amazon EC2 to ElastiCache (re:Invent 2019)

**Scaling a Fantasy Sports Platform with Amazon ElastiCache & Amazon Aurora STP11 (re:Invent 2018)**

Dream11 is India's leading sports-tech startup. It has a growing base of 40 million+ users playing multiple sports, including fantasy cricket, football, and basketball, and it currently serves one million concurrent users, who produce three million requests per minute under a 50-millisecond

response time. In this talk, Dream11 CTO Amit Sharma explains how the company uses Amazon Aurora and Amazon ElastiCache to handle flash traffic, which can triple within a 30-second response window. Sharma also talks about scaling transactions without locking, and he shares the steps for handling flash traffic—thereby serving five million daily active users. Complete Title: AWS re:Invent 2018: Scaling a Fantasy Sports Platform with Amazon ElastiCache & Amazon Aurora (STP11)

Scaling a Fantasy Sports Platform with Amazon ElastiCache & Amazon Aurora STP11 (re:Invent 2018)

**Reliable & Scalable Redis OSS in the Cloud with Amazon ElastiCache (re:Invent 2018)**

This session covers the features and enhancements in our Redis OSS-compatible service, Amazon ElastiCache for Redis OSS. We cover key features, such as Redis OSS 5, scalability and performance improvements, security and compliance, and much more. We also discuss upcoming features and customer case studies.

Reliable & Scalable Redis OSS in the Cloud with Amazon ElastiCache (re:Invent 2018)

**ElastiCache Deep Dive: Design Patterns for In-Memory Data Stores (re:Invent 2018)**

In this session, we provide a behind the scenes peek to learn about the design and architecture of Amazon ElastiCache. See common design patterns with our Redis OSS and Memcached offerings and how customers use them for in-memory data processing to reduce latency and improve application throughput. We review ElastiCache best practices, design patterns, and anti-patterns.

ElastiCache Deep Dive: Design Patterns for In-Memory Data Stores (re:Invent 2018)

**DAT305—Amazon ElastiCache Deep Dive (re:Invent 2017)**

Look behind the scenes to learn about Amazon ElastiCache's design and architecture. See common design patterns with our Memcached and Redis OSS offerings and how customers have used them for in-memory operations to reduce latency and improve application throughput. During this video, we review ElastiCache best practices, design patterns, and anti-patterns.

The video introduces the following:

- ElastiCache for Redis OSS online resharding
- ElastiCache security and encryption
- ElastiCache for Redis OSS version 3.2.10

DAT305—Amazon ElastiCache Deep Dive (re:Invent 2017)

**DAT306—Amazon ElastiCache Deep Dive (re:Invent 2016)**

Look behind the scenes to learn about Amazon ElastiCache's design and architecture. See common design patterns with our Memcached and Redis OSS offerings and how customers have used them for in-memory operations to reduce latency and improve application throughput. During this session, we review ElastiCache best practices, design patterns, and anti-patterns.

DAT306—Amazon ElastiCache Deep Dive (re:Invent 2016)

**DAT317—How IFTTT uses ElastiCache for Redis OSS to Predict Events (re:Invent 2016)**

IFTTT is a free service that empowers people to do more with the services they love, from automating simple tasks to transforming how someone interacts with and controls their home. IFTTT uses ElastiCache for Redis OSS to store transaction run history and schedule predictions as well as indexes for log documents on Amazon S3. View this session to learn how the scripting power of Lua and the data types of Redis OSS allowed people to accomplish something they wouldn't have been able to elsewhere.

DAT317—How IFTTT uses ElastiCache for Redis OSS to Predict Events (re:Invent 2016)

**DAT407—Amazon ElastiCache Deep Dive (re:Invent 2015)**

Peek behind the scenes to learn about Amazon ElastiCache's design and architecture. See common design patterns of our Memcached and Redis OSS offerings and how customers have used them for in-memory operations and achieved improved latency and throughput for applications. During this session, we review best practices, design patterns, and anti-patterns related to Amazon ElastiCache.

DAT407—Amazon ElastiCache Deep Dive (re:Invent 2015)

**SDD402—Amazon ElastiCache Deep Dive (re:Invent 2014)**

In this video, we examine common caching use cases, the ElastiCache for Memcached and ElastiCache for Redis OSS engines, patterns that help you determine which engine is better for your needs, consistent hashing, and more as means to building fast, scalable applications. Frank Wiebe, Principal Scientist at Adobe, details how Adobe uses Amazon ElastiCache to improve customer experience and scale their business.

DAT402—Amazon ElastiCache Deep Dive (re:Invent 2014)

**DAT307—Deep Dive into Amazon ElastiCache Architecture and Design Patterns (re:Invent 2013)**

In this video, we examine caching, caching strategies, scaling out, monitoring. We also compare the ElastiCache for Memcached and ElastiCache for Redis OSS engines. During this session, also we review best practices and design patterns related to Amazon ElastiCache.

DAT307 - Deep Dive into Amazon ElastiCache Architecture and Design Patterns (AWS re:Invent 2013).

# Managing nodes in ElastiCache

A node is the smallest building block of an Amazon ElastiCache deployment. It is a fixed-size chunk of secure, network-attached RAM. Each node runs the engine that was chosen when the cluster was created or last modified. Each node has its own Domain Name Service (DNS) name and port. Multiple types of ElastiCache nodes are supported, each with varying amounts of associated memory and computational power.

For a more detailed discussion of which node size to use, see Choosing your node size.

Generally speaking, due to its support for sharding, Valkey or Redis OSS (cluster mode enabled) deployments have a number of smaller nodes. In contrast, Valkey or Redis OSS (cluster mode disabled) deployments have fewer, larger nodes in a cluster. For a more detailed discussion of which node size to use, see Choosing your node size.

**Topics**


- Viewing ElastiCache Node Status
- Valkey or Redis OSS nodes and shards
- Connecting to nodes
- Supported node types
- Rebooting nodes
- Replacing nodes (Valkey and Redis OSS)
- Replacing nodes (Memcached)
- Reserved nodes
- Migrating previous generation nodes


Some important operations involving nodes are the following:

- Adding nodes to an ElastiCache cluster
- Removing nodes from an ElastiCache cluster
- Scaling ElastiCache
- Finding connection endpoints in ElastiCache
- Automatically identify nodes in your cluster (Memcached)

## Viewing ElastiCache Node Status

Using the ElastiCache console, you can quickly access the status of your ElastiCache node. The status of an ElastiCache node indicates the health of the node. You can use the following procedures to view the ElastiCache node status in the Amazon ElastiCache console, the AWS CLI command, or the API operation.

The possible status values for ElastiCache nodes are in the following table. This table also shows if you will be billed for the ElastiCache node.

| Type | Billed | Description |
| --- | --- | --- |
| `available` | Billed | The ElastiCache node is healthy and available. |
| `creating` | Not billed | The ElastiCache node is being created. The Node is inaccessible while it is being created. |
| `deleting` | Not billed | The ElastiCache node is being deleted. |
| `modifying` | Billed | The ElastiCache node is being modified because of a customer request to modify the node. |
| `updating` | Billed | An **Updating** state indicates one or more of the following is true of the Amazon ElastiCache node: |

| Type | Billed | Description |
| --- | --- | --- |
| | | • The ElastiCache node is being patched as part of the service udpate. For more information on the service updates, refer to the Amazon ElastiCache Managed Maintenance and Service Updates Help Page.<br>• The VPC security groups are updating for the ElastiCac he Cluster.<br>• The ElastiCache cluster is being scaled up or scaled down.<br>• The log delivery configura tions are being modified for the ElastiCache Cluster.<br>• A delete operation for the ElastiCache node is pending.<br>• The ElastiCache password is being updated/r otated using AWS Secrets Manager. |
| `rebooting cluster nodes` | Billed | The ElastiCache node is being rebooted because of a customer request or an Amazon ElastiCache process that requires the rebooting of the node. |

| Type | Billed | Description |
| --- | --- | --- |
| `incompatible_parameters` | Not billed | Amazon ElastiCache can't start the node because the parameters specified in the node's parameter group aren't compatible with the node. Either revert the parameter changes or make them compatible with the node to regain access to your node. For more information about the incompatible parameters, check the Events list for the ElastiCache node. |
| `incompatible_network` | Not billed | An incompatible-network state indicates one or more of the following is true of the Amazon ElastiCache node:<br>• There are no available IP addresses in the subnet that the ElastiCache node was launched into.<br>• The subnet mentioned in the ElastiCache subnet group no longer exists in the Amazon Virtual Private Cloud (Amazon VPC). |

| Type | Billed | Description |
| --- | --- | --- |
| `restore_failed` | Not billed | A restore-failed state indicates one of the following is true of the Amazon ElastiCache node:<br>• Replacements of node failed due to Insufficient instance capacity repeatedly. This typically happens when running previous generation nodes that are end-of-life. However, it could also happen with replacement of current generation nodes when AWS does not have enough on-demand capacity to fulfill your request in the specified Availability Zone. For more information on fixing or removing these nodes, see Migrating previous generation nodes.<br>• The specified RDB snapshot failed to restore.<br>• The AWS account for the ElastiCache cluster has been suspended.<br>• The node failed and could not be recovered. |
| `snapshotting` | Billed | ElastiCache is creating a snapshot of the Valkey or Redis OSS node. |

### Viewing ElastiCache Node Status with the console

To view the status of an ElastiCache Node with the console:

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Redis OSS Clusters** or **Memcached Clusters**. The **Caches page** appears with the list of ElastiCache Nodes. For each node, the status value is displayed.
3. You can then navigate to the **Service Updates** tab for the cache to display the list of service Updates applicable to the cache.

### Viewing ElastiCache Node Status with the AWS CLI

To view ElastiCache node and its status information by using the AWS CLI, use the `describe-cache-cluster` command. For example, the following AWS CLI command displays each ElastiCache node.

```
aws elasticache describe-cache-clusters
```

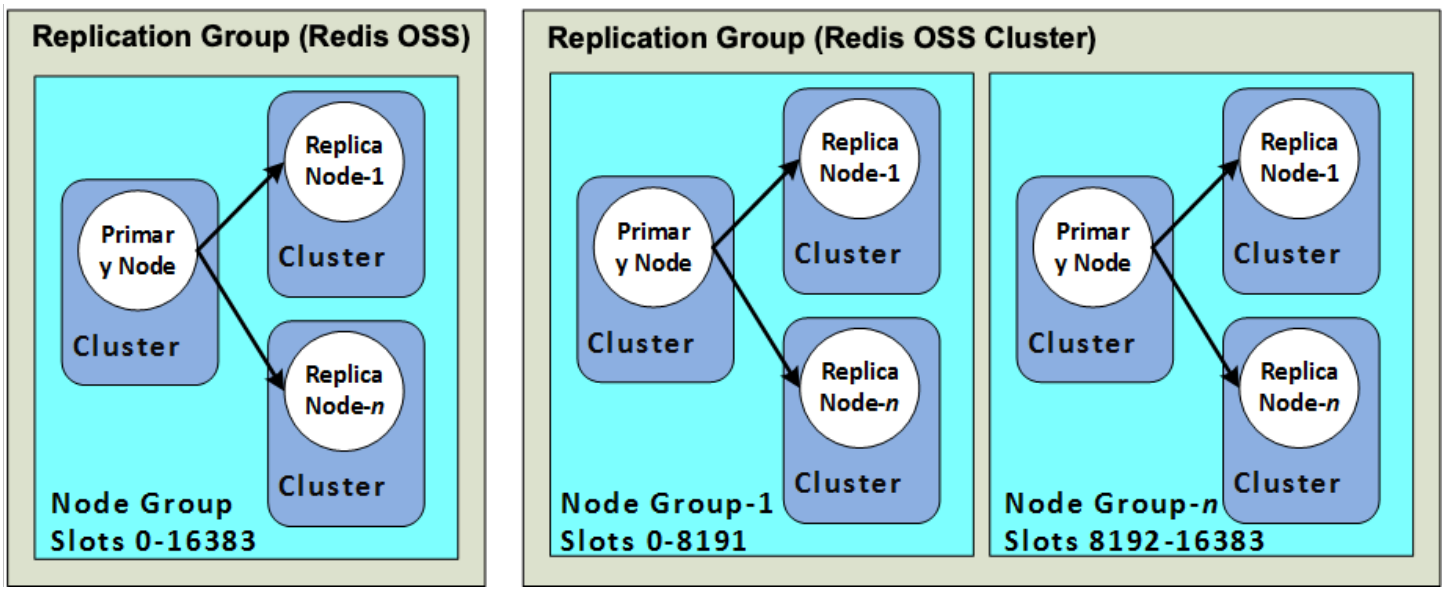
### Viewing ElastiCache Node Status through the API

To view the status of the ElastiCache node using the Amazon ElastiCache API, call the `DescribeCacheClusteroperation` with the `ShowCacheNodeInfo` flag to retrieve information about the individual cache nodes.

## Valkey or Redis OSS nodes and shards

A shard (in the API and CLI, a node group) is a hierarchical arrangement of nodes, each wrapped in a cluster. Shards support replication. Within a shard, one node functions as the read/write primary node. All the other nodes in a shard function as read-only replicas of the primary node. Valkey and Redis OSS support multiple shards within a cluster (in the API and CLI, a replication group). This support enables partitioning your data in a Valkey or Redis OSS (cluster mode enabled) cluster.

The following diagram illustrates the differences between a Valkey or Redis OSS (cluster mode disabled) cluster and a Valkey or Redis OSS (cluster mode enabled) cluster.




Valkey or Redis OSS (cluster mode enabled) clusters support replication via shards. The API operation DescribeReplicationGroups (CLI: describe-replication-groups) lists the node groups with the member nodes, the node's role within the node group, and also other information.

When you create a Valkey or Redis OSS cluster, you specify whether you want to create a cluster with clustering enabled. Valkey or Redis OSS (cluster mode disabled) clusters never have more than one shard, which can be scaled horizontally by adding (up to a total of five) or deleting read replica nodes. For more information, see High availability using replication groups, Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled) or Deleting a read replica for Valkey or Redis OSS (Cluster Mode Disabled). Valkey or Redis OSS (cluster mode disabled) clusters can also scale vertically by changing node types. For more information, see Scaling replica nodes for Valkey or Redis OSS (Cluster Mode Disabled).

The node or shard limit can be increased to a maximum of 500 per cluster if the engine is Valkey or Redis OSS version 5.0.6 or higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group.

For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

After a Valkey or Redis OSS (cluster mode enabled) cluster is created, it can be altered (scaled in or out). For more information, see Scaling ElastiCache and Replacing nodes (Valkey and Redis OSS).

When you create a new cluster, you can seed it with data from the old cluster so it doesn't start out empty. This approach works only if the cluster group has the same number of shards as the old cluster. Doing this can be helpful if you need change your node type or engine version. For more information, see Taking manual backups and Restoring from a backup into a new cache.

# Connecting to nodes

## Connecting to Valkey or Redis OSS nodes

Before attempting to connect to the Valkey or Redis OSS nodes in your cluster, you must have the endpoints for the nodes. To find the endpoints, see the following:

- Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding Endpoints for a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (Console)
- Finding Endpoints (AWS CLI)
- Finding Endpoints (ElastiCache API)

In the following example, you use the *valkey-cli* utility to connect to a cluster that is running Valkey or Redis OSS.

> **Note**
>
> For more information about available commands, see the Commands webpage.

### To connect to a Valkey or Redis OSS cluster using the *valkey-cli*

1. Connect to your Amazon EC2 instance using the connection utility of your choice.

   > **Note**
   >
   > For instructions on how to connect to an Amazon EC2 instance, see the Amazon EC2 Getting Started Guide.

2. To build `valkey-cli`, download and install the GNU Compiler Collection (`gcc`). At the command prompt of your EC2 instance, enter the following command and enter `y` at the confirmation prompt.

   ```
   sudo yum install gcc
   ```

   Output similar to the following appears.

   ```
   Loaded plugins: priorities, security, update-motd, upgrade-helper
   ```

```
Setting up Install Process
Resolving Dependencies
--> Running transaction check

...(output omitted)...

Total download size: 27 M
Installed size: 53 M
Is this ok [y/N]: y
Downloading Packages:
(1/11): binutils-2.22.52.0.1-10.36.amzn1.x86_64.rpm      | 5.2 MB     00:00
(2/11): cpp46-4.6.3-2.67.amzn1.x86_64.rpm                | 4.8 MB     00:00
(3/11): gcc-4.6.3-3.10.amzn1.noarch.rpm                  | 2.8 kB     00:00

...(output omitted)...

Complete!
```

3. Download and compile the *valkey-cli* utility. This utility is included in the Valkey software distribution. At the command prompt of your EC2 instance, type the following commands:

> **Note**
>
> For Ubuntu systems, before running `make`, run `make distclean`.

```
wget -O valkey-8.0.0.tar.gz https://github.com/valkey-io/valkey/archive/refs/
tags/8.0.0.tar.gz
tar xvzf valkey-8.0.0.tar.gz
cd valkey-8.0.0
make distclean      # ubuntu systems only
make
```

4. At the command prompt of your EC2 instance, type the following command.

```
src/valkey-cli -c -h mycachecluster.eaogs8.0001.usw2.cache.amazonaws.com -p 6379
```

A Valkey or Redis OSS command prompt similar to the following appears.

```
redis mycachecluster.eaogs8.0001.usw2.cache.amazonaws.com 6379>
```

5. Test the connection by running Valkey or Redis OSS commands.

   You are now connected to the cluster and can run Valkey or Redis OSS commands. The following are some example commands with their Valkey or Redis OSS responses.

```
set a "hello"          // Set key "a" with a string value and no expiration
OK
get a                  // Get value for key "a"
"hello"
get b                  // Get value for key "b" results in miss
(nil)
set b "Good-bye" EX 5  // Set key "b" with a string value and a 5 second expiration
get b
"Good-bye"
                   // wait 5 seconds
get b
(nil)                  // key has expired, nothing returned
quit                   // Exit from valkey-cli
```

For connecting to nodes or clusters which have Secure Sockets Layer (SSL) encryption (in-transit enabled), see ElastiCache in-transit encryption (TLS).

## Connecting to Memcached nodes

Before attempting to connect to your Memcached cluster, you must have the endpoints for the nodes. To find the endpoints, see the following:

- Finding a Cluster's Endpoints (Console) (Memcached)
- Finding Endpoints (AWS CLI)
- Finding Endpoints (ElastiCache API)

In the following example, you use the *telnet* utility to connect to a node that is running Memcached.

> **Note**
>
> For more information about Memcached and available Memcached commands, see the Memcached website.

**To connect to a node using *telnet***

1. Connect to your Amazon EC2 instance by using the connection utility of your choice.

   > **Note**
   >
   > For instructions on how to connect to an Amazon EC2 instance, see the Amazon EC2 Getting Started Guide.

2. Download and install the *telnet* utility on your Amazon EC2 instance. At the command prompt of your Amazon EC2 instance, type the following command and type *y* at the command prompt.

   ```
   sudo yum install telnet
   ```

   Output similar to the following appears.

   ```
   Loaded plugins: priorities, security, update-motd, upgrade-helper
   Setting up Install Process
   Resolving Dependencies
   --> Running transaction check

   ...(output omitted)...

   Total download size: 63 k
   Installed size: 109 k
   Is this ok [y/N]: y
   Downloading Packages:
   telnet-0.17-47.7.amzn1.x86_64.rpm                          |  63 kB     00:00

   ...(output omitted)...
   ```

```
Complete!
```

3. At the command prompt of your Amazon EC2 instance, type the following command, substituting the endpoint of your node for the one shown in this example.

```
telnet mycachecluster.eaogs8.0001.usw2.cache.amazonaws.com 11211
```

Output similar to the following appears.

```
Trying 128.0.0.1...
Connected to mycachecluster.eaogs8.0001.usw2.cache.amazonaws.com.
Escape character is '^]'.
>
```

4. Test the connection by running Memcached commands.

You are now connected to a node, and you can run Memcached commands. The following is an example.

```
set a 0 0 5      // Set key "a" with no expiration and 5 byte value
hello            // Set value as "hello"
STORED
get a            // Get value for key "a"
VALUE a 0 5
hello
END
get b            // Get value for key "b" results in miss
END
>
```

# Supported node types

ElastiCache supports the following node types. Generally speaking, the latest generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For performance details for each node type, see Amazon EC2 Instance Types.

For information on which node size to use, see Choosing your node size.

> **Note**
>
> Instance types with burstable network performance use a network I/O credit mechanism to burst beyond their baseline bandwidth on a best-effort basis.

**Topics**


- Current Generation (Memcached)
- Current Generation (Valkey)
- Current Generation (Redis OSS)
- Burstable Performance Nodes
- Previous Generation
- Supported node types by AWS Region
- Related Information


## Current Generation (Memcached)

### General

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.m8g.large | 1.6.6 | 0.937 | 12.5 | 6.79 |
| cache.m8g.xlarge | 1.6.6 | 1.875 | 12.5 | 14.18 |

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.m8g .2xlarge | 1.6.6 | 3.75 | 15 | 29.68 |
| cache.m8g .4xlarge | 1.6.6 | 7.5 | 15 | 61.33 |
| cache.m8g .8xlarge | 1.6.6 | 15 | N/A | 124.65 |
| cache.m8g .12xlarge | 1.6.6 | 22.5 | N/A | 187.96 |
| cache.m8g .16xlarge | 1.6.6 | 30 | N/A | 251.27 |
| cache.m7g.large | 1.5.16 | 0.937 | 12.5 | 6.38 |
| cache.m7g.xlarge | 1.5.16 | 1.876 | 12.5 | 12.93 |
| cache.m7g .2xlarge | 1.5.16 | 3.75 | 15 | 26.04 |
| cache.m7g .4xlarge | 1.5.16 | 7.5 | 15 | 52.26 |
| cache.m7g .8xlarge | 1.5.16 | 15 | N/A | 103.68 |
| cache.m7g .12xlarge | 1.5.16 | 22.5 | N/A | 157.12 |
| cache.m7g .16xlarge | 1.5.16 | 30 | N/A | 209.55 |
| cache.m6g.large | 1.5.16 | 0.75 | 10.0 | 6.38 |

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.m6g.xlarge | 1.5.16 | 1.25 | 10.0 | 12.93 |
| cache.m6g .2xlarge | 1.5.16 | 2.5 | 10.0 | 26.04 |
| cache.m6g .4xlarge | 1.5.16 | 5.0 | 10.0 | 52.26 |
| cache.m6g .8xlarge | 1.5.16 | 12 | N/A | 103.68 |
| cache.m6g .12xlarge | 1.5.16 | 20 | N/A | 157.12 |
| cache.m6g .16xlarge | 1.5.16 | 25 | N/A | 209.55 |
| cache.m5.large | 1.5.16 | 0.75 | 10.0 | 6.38 |
| cache.m5.xlarge | 1.5.16 | 1.25 | 10.0 | 12.93 |
| cache.m5.2xlarge | 1.5.16 | 2.5 | 10.0 | 26.04 |
| cache.m5.4xlarge | 1.5.16 | 5.0 | 10.0 | 52.26 |
| cache.m5. 12xlarge | 1.5.16 | 12 | N/A | 157.12 |
| cache.m5. 24xlarge | 1.5.16 | 25 | N/A | 314.32 |
| cache.m4.large | 1.5.16 | 0.45 | 1.2 | 6.42 |
| cache.m4.xlarge | 1.5.16 | 0.75 | 2.8 | 14.28 |

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.m4.2xlarge | 1.5.16 | 1.0 | 10.0 | 29.70 |
| cache.m4.4xlarge | 1.5.16 | 2.0 | 10.0 | 60.78 |
| cache.m4.10xlarge | 1.5.16 | 5.0 | 10.0 | 154.64 |
| cache.t4g.micro | 1.5.16 | 0.064 | 5.0 | 0.50 |
| cache.t4g.small | 1.5.16 | 0.128 | 5.0 | 1.37 |
| cache.t4g.medium | 1.5.16 | 0.256 | 5.0 | 3.09 |
| cache.t3.micro | 1.5.16 | 0.064 | 5.0 | 0.50 |
| cache.t3.small | 1.5.16 | 0.128 | 5.0 | 1.37 |
| cache.t3.medium | 1.5.16 | 0.256 | 5.0 | 3.09 |

**Memory optimized**

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.r8g.large | 1.6.6 | 0.937 | 12.5 | 14.18 |
| cache.r8g.xlarge | 1.6.6 | 1.875 | 12.5 | 29.68 |
| cache.r8g.2xlarge | 1.6.6 | 3.75 | 15 | 61.33 |
| cache.r8g.4xlarge | 1.6.6 | 7.5 | 15 | 124.65 |

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.r8g.8xlarge | 1.6.6 | 15 | N/A | 251.27 |
| cache.r8g .12xlarge | 1.6.6 | 22.5 | N/A | 377.9 |
| cache.r8g .16xlarge | 1.6.6 | 30 | N/A | 504.52 |
| cache.r7g.large | 1.5.16 | 0.937 | 12.5 | 13.07 |
| cache.r7g.xlarge | 1.5.16 | 1.876 | 12.5 | 26.32 |
| cache.r7g.2xlarge | 1.5.16 | 3.75 | 15 | 52.82 |
| cache.r7g.4xlarge | 1.5.16 | 7.5 | 15 | 105.81 |
| cache.r7g.8xlarge | 1.5.16 | 15 | N/A | 209.55 |
| cache.r7g .12xlarge | 1.5.16 | 22.5 | N/A | 317.77 |
| cache.r7g .16xlarge | 1.5.16 | 30 | N/A | 419.09 |
| cache.r6g.large | 1.5.16 | 0.75 | 10.0 | 13.07 |
| cache.r6g.xlarge | 1.5.16 | 1.25 | 10.0 | 26.32 |
| cache.r6g.2xlarge | 1.5.16 | 2.5 | 10.0 | 52.82 |
| cache.r6g.4xlarge | 1.5.16 | 5.0 | 10.0 | 105.81 |
| cache.r6g.8xlarge | 1.5.16 | 12 | N/A | 209.55 |
| cache.r6g .12xlarge | 1.5.16 | 20 | N/A | 317.77 |

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.r6g .16xlarge | 1.5.16 | 25 | N/A | 419.09 |
| cache.r5.large | 1.5.16 | 0.75 | 10.0 | 13.07 |
| cache.r5.xlarge | 1.5.16 | 1.25 | 10.0 | 26.32 |
| cache.r5.2xlarge | 1.5.16 | 2.5 | 10.0 | 52.82 |
| cache.r5.4xlarge | 1.5.16 | 5.0 | 10.0 | 105.81 |
| cache.r5.12xlarge | 1.5.16 | 20 | N/A | 317.77 |
| cache.r5.24xlarge | 1.5.16 | 25 | N/A | 635.61 |
| cache.r4.large | 1.5.16 | 0.75 | 10.0 | 12.30 |
| cache.r4.xlarge | 1.5.16 | 1.25 | 10.0 | 25.05 |
| cache.r4.2xlarge | 1.5.16 | 2.5 | 10.0 | 50.47 |
| cache.r4.4xlarge | 1.5.16 | 5.0 | 10.0 | 101.38 |
| cache.r4.8xlarge | 1.5.16 | 12 | N/A | 203.26 |
| cache.r4.16xlarge | 1.5.16 | 25 | N/A | 407.00 |

**Network optimized**

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.c8gn.large | 1.6.6 | 6.25 | 30 | 3.1 |
| cache.c8gn.xlarge | 1.6.6 | 12.5 | 40 | 6.79 |
| cache.c8gn.2xlarge | 1.6.6 | 25 | 50 | 14.18 |
| cache.c8gn.4xlarge | 1.6.6 | 50 | N/A | 29.68 |
| cache.c8gn.8xlarge | 1.6.6 | 100 | N/A | 61.33 |
| cache.c8gn.12xlarge | 1.6.6 | 150 | N/A | 92.99 |
| cache.c8gn.16xlarge | 1.6.6 | 200 | N/A | 124.65 |
| cache.c7gn.large | 1.6.6 | 6.25 | 30 | 3.09 |
| cache.c7gn.xlarge | 1.6.6 | 12.5 | 40 | 6.38 |
| cache.c7gn.2xlarge | 1.6.6 | 25 | 50 | 12.94 |
| cache.c7gn.4xlarge | 1.6.6 | 50 | N/A | 26.05 |
| cache.c7gn.8xlarge | 1.6.6 | 100 | N/A | 52.26 |
| cache.c7gn.12xlarge | 1.6.6 | 150 | N/A | 78.56 |

| Instance type | Minimum supported Memcached version | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|
| cache.c7gn.16xlarge | 1.6.6 | 200 | N/A | 105.81 |

## Current Generation (Valkey)

### General

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m8g.large | 7.2 | Yes | No | Yes | 0.937 | 12.5 | 6.79 |
| cache.m8g.xlarge | 7.2 | Yes | Yes | Yes | 1.875 | 12.5 | 14.18 |
| cache.m8g.2xlarge | 7.2 | Yes | Yes | Yes | 3.75 | 15 | 29.68 |
| cache.m8g.4xlarge | 7.2 | Yes | Yes | Yes | 7.5 | 15 | 61.33 |
| cache.m8g.8xlarge | 7.2 | Yes | Yes | Yes | 15 | N/A | 124.65 |
| cache.m8g.12xlarge | 7.2 | Yes | Yes | Yes | 22.5 | N/A | 187.96 |
| cache.m8g.16xlarge | 7.2 | Yes | Yes | Yes | 30 | N/A | 251.27 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m7g.large | 7.2 | Yes | No | Yes | 0.937 | 12.5 | 6.38 |
| cache.m7g.xlarge | 7.2 | Yes | Yes | Yes | 1.876 | 12.5 | 12.93 |
| cache.m7g.2xlarge | 7.2 | Yes | Yes | Yes | 3.75 | 15 | 26.04 |
| cache.m7g.4xlarge | 7.2 | Yes | Yes | Yes | 7.5 | 15 | 52.26 |
| cache.m7g.8xlarge | 7.2 | Yes | Yes | Yes | 15 | N/A | 103.68 |
| cache.m7g.12xlarge | 7.2 | Yes | Yes | Yes | 22.5 | N/A | 157.12 |
| cache.m7g.16xlarge | 7.2 | Yes | Yes | Yes | 30 | N/A | 209.55 |
| cache.m6g.large | 7.2 | No | No | No | 0.75 | 10.0 | 6.38 |
| cache.m6g.xlarge | 7.2 | Yes | Yes | Yes | 1.25 | 10.0 | 12.93 |
| cache.m6g.2xlarge | 7.2 | Yes | Yes | Yes | 2.5 | 10.0 | 26.04 |
| cache.m6g.4xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10.0 | 52.26 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m6g.8xlarge | 7.2 | Yes | Yes | Yes | 12 | N/A | 103.68 |
| cache.m6g.12xlarge | 7.2 | Yes | Yes | Yes | 20 | N/A | 157.12 |
| cache.m6g.16xlarge | 7.2 | Yes | Yes | Yes | 25 | N/A | 209.55 |
| cache.m5.large | 7.2 | No | No | No | 0.75 | 10.0 | 6.38 |
| cache.m5.xlarge | 7.2 | Yes | No | No | 1.25 | 10.0 | 12.93 |
| cache.m5.2xlarge | 7.2 | Yes | Yes | Yes | 2.5 | 10.0 | 26.04 |
| cache.m5.4xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10.0 | 52.26 |
| cache.m5.12xlarge | 7.2 | Yes | Yes | Yes | 12 | N/A | 157.12 |
| cache.m5.24xlarge | 7.2 | Yes | Yes | Yes | 25 | N/A | 314.32 |
| cache.m4.large | 7.2 | No | No | No | 0.45 | 1.2 | 6.42 |
| cache.m4.xlarge | 7.2 | Yes | No | No | 0.75 | 2.8 | 14.28 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m4.2xlarge | 7.2 | Yes | Yes | Yes | 1.0 | 10.0 | 29.70 |
| cache.m4.4xlarge | 7.2 | Yes | Yes | Yes | 2.0 | 10.0 | 60.78 |
| cache.m4.10xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10.0 | 154.64 |
| cache.t4g.micro | 7.2 | No | No | No | 0.064 | 5.0 | 0.50 |
| cache.t4g.small | 7.2 | No | No | No | 0.128 | 5.0 | 1.37 |
| cache.t4g.medium | 7.2 | No | No | No | 0.256 | 5.0 | 3.09 |
| cache.t3.micro | 7.2 | No | No | No | 0.064 | 5.0 | 0.50 |
| cache.t3.small | 7.2 | No | No | No | 0.128 | 5.0 | 1.37 |
| cache.t3.medium | 7.2 | No | No | No | 0.256 | 5.0 | 3.09 |

### Memory optimized

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.r8g.large | 7.2 | Yes | No | Yes | 0.937 | 12.5 | 14.18 |
| cache.r8g.xlarge | 7.2 | Yes | Yes | Yes | 1.875 | 12.5 | 29.68 |
| cache.r8g.2xlarge | 7.2 | Yes | Yes | Yes | 3.75 | 15 | 61.33 |
| cache.r8g.4xlarge | 7.2 | Yes | Yes | Yes | 7.5 | 15 | 124.65 |
| cache.r8g.8xlarge | 7.2 | Yes | Yes | Yes | 15 | N/A | 251.27 |
| cache.r8g.12xlarge | 7.2 | Yes | Yes | Yes | 22.5 | N/A | 377.9 |
| cache.r8g.16xlarge | 7.2 | Yes | Yes | Yes | 30 | N/A | 504.52 |
| cache.r7g.large | 7.2 | Yes | No | Yes | 0.937 | 12.5 | 13.07 |
| cache.r7g.xlarge | 7.2 | Yes | Yes | Yes | 1.876 | 12.5 | 26.32 |
| cache.r7g.2xlarge | 7.2 | Yes | Yes | Yes | 3.75 | 15 | 52.82 |
| cache.r7g.4xlarge | 7.2 | Yes | Yes | Yes | 7.5 | 15 | 105.81 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.r7g.8xlarge | 7.2 | Yes | Yes | Yes | 15 | N/A | 209.55 |
| cache.r7g.12xlarge | 7.2 | Yes | Yes | Yes | 22.5 | N/A | 317.77 |
| cache.r7g.16xlarge | 7.2 | Yes | Yes | Yes | 30 | N/A | 419.09 |
| cache.r6g.large | 7.2 | No | No | No | 0.75 | 10.0 | 13.07 |
| cache.r6g.xlarge | 7.2 | Yes | Yes | Yes | 1.25 | 10.0 | 26.32 |
| cache.r6g.2xlarge | 7.2 | Yes | Yes | Yes | 2.5 | 10.0 | 52.82 |
| cache.r6g.4xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10.0 | 105.81 |
| cache.r6g.8xlarge | 7.2 | Yes | Yes | Yes | 12 | N/A | 209.55 |
| cache.r6g.12xlarge | 7.2 | Yes | Yes | Yes | 20 | N/A | 317.77 |
| cache.r6g.16xlarge | 7.2 | Yes | Yes | Yes | 25 | N/A | 419.09 |
| cache.r5.large | 7.2 | No | No | No | 0.75 | 10.0 | 13.07 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.r5.xlarge | 7.2 | Yes | No | No | 1.25 | 10.0 | 26.32 |
| cache.r5.2xlarge | 7.2 | Yes | Yes | Yes | 2.5 | 10.0 | 52.82 |
| cache.r5.4xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10.0 | 105.81 |
| cache.r5.12xlarge | 7.2 | Yes | Yes | Yes | 12 | N/A | 317.77 |
| cache.r5.24xlarge | 7.2 | Yes | Yes | Yes | 25 | N/A | 635.61 |
| cache.r4.large | 7.2 | No | No | No | 0.75 | 10.0 | 12.30 |
| cache.r4.xlarge | 7.2 | Yes | No | No | 1.25 | 10.0 | 25.05 |
| cache.r4.2xlarge | 7.2 | Yes | Yes | Yes | 2.5 | 10.0 | 50.47 |
| cache.r4.4xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10.0 | 101.38 |
| cache.r4.8xlarge | 7.2 | Yes | Yes | Yes | 12 | N/A | 203.26 |
| cache.r4.16xlarge | 7.2 | Yes | Yes | Yes | 25 | N/A | 407.00 |

### Memory optimized with data tiering

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) | SSD (GiB) |
|---|---|---|---|---|---|---|---|---|
| cache.r8gd.xlarge | 7.2 | Yes | No | No | 1.875 | 12.5 | 29.68 | 99.33 |
| cache.r8gd.2xlarge | 7.2 | Yes | Yes | Yes | 3.75 | 15 | 61.33 | 198.65 |
| cache.r8gd.4xlarge | 7.2 | Yes | Yes | Yes | 7.5 | 15 | 124.65 | 398.14 |
| cache.r8gd.8xlarge | 7.2 | Yes | Yes | Yes | 15 | N/A | 251.27 | 796.28 |
| cache.r8gd.12xlarge | 7.2 | Yes | Yes | Yes | 22.5 | N/A | 377.9 | 1194.42 |
| cache.r8gd.16xlarge | 7.2 | Yes | Yes | Yes | 30 | N/A | 504.52 | 1592.56 |
| cache.r7gd.xlarge | 7.2 | Yes | No | No | 1.875 | 12.5 | 29.68 | 99.33 |
| cache.r7gd.2xlarge | 7.2 | Yes | Yes | Yes | 3.75 | 15 | 61.33 | 198.65 |
| cache.r7gd.4xlarge | 7.2 | Yes | Yes | Yes | 7.5 | 15 | 124.65 | 398.14 |
| cache.r7gd.8xlarge | 7.2 | Yes | Yes | Yes | 15 | N/A | 251.27 | 796.28 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) | SSD (GiB) |
|---|---|---|---|---|---|---|---|---|
| cache.r7gd.12xlarge | 7.2 | Yes | Yes | Yes | 22.5 | N/A | 377.9 | 1194.42 |
| cache.r7gd.16xlarge | 7.2 | Yes | Yes | Yes | 30 | N/A | 504.52 | 1592.56 |
| cache.r6gd.xlarge | 7.2 | Yes | No | No | 1.25 | 10 | 26.32 | 99.33 |
| cache.r6gd.2xlarge | 7.2 | Yes | Yes | Yes | 2.5 | 10 | 52.82 | 199.07 |
| cache.r6gd.4xlarge | 7.2 | Yes | Yes | Yes | 5.0 | 10 | 105.81 | 398.14 |
| cache.r6gd.8xlarge | 7.2 | Yes | Yes | Yes | 12 | N/A | 209.55 | 796.28 |
| cache.r6gd.12xlarge | 7.2 | Yes | Yes | Yes | 20 | N/A | 317.77 | 1194.42 |
| cache.r6gd.16xlarge | 7.2 | Yes | Yes | Yes | 25 | N/A | 419.09 | 1592.56 |

### Network optimized

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.c8gn.large | 7.2 | No | No | No | 6.25 | 30 | 3.1 |
| cache.c8gn.xlarge | 7.2 | Yes | Yes | Yes | 12.5 | 40 | 6.79 |
| cache.c8gn.2xlarge | 7.2 | Yes | Yes | Yes | 25 | 50 | 14.18 |
| cache.c8gn.4xlarge | 7.2 | Yes | Yes | Yes | 50 | N/A | 29.68 |
| cache.c8gn.8xlarge | 7.2 | Yes | Yes | Yes | 100 | N/A | 61.33 |
| cache.c8gn.12xlarge | 7.2 | Yes | Yes | Yes | 150 | N/A | 92.99 |
| cache.c8gn.16xlarge | 7.2 | Yes | Yes | Yes | 200 | N/A | 124.65 |
| cache.c7gn.large | 7.2 | No | No | No | 6.25 | 30 | 3.09 |
| cache.c7gn.xlarge | 7.2 | Yes | Yes | Yes | 12.5 | 40 | 6.38 |
| cache.c7gn.2xlarge | 7.2 | Yes | Yes | Yes | 25 | 50 | 12.94 |
| cache.c7gn.4xlarge | 7.2 | Yes | Yes | Yes | 50 | N/A | 26.05 |

| Instance type | Minimum supported Valkey version | Enhanced I/O | TLS Offloading | Enhanced I/O Multiplexing | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.c7gn.8xlarge | 7.2 | Yes | Yes | Yes | 100 | N/A | 52.26 |
| cache.c7gn.12xlarge | 7.2 | Yes | Yes | Yes | 150 | N/A | 78.56 |
| cache.c7gn.16xlarge | 7.2 | Yes | Yes | Yes | 200 | N/A | 105.81 |

## Current Generation (Redis OSS)

### General

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m7g.large | 6.2 | Yes | No | Yes | 0.937 | 12.5 | 6.38 |
| cache.m7g.xlarge | 6.2 | Yes | Yes | Yes | 1.876 | 12.5 | 12.93 |
| cache.m7g.2xlarge | 6.2 | Yes | Yes | Yes | 3.75 | 15 | 26.04 |

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m7g.4xlarge | 6.2 | Yes | Yes | Yes | 7.5 | 15 | 52.26 |
| cache.m7g.8xlarge | 6.2 | Yes | Yes | Yes | 15 | N/A | 103.68 |
| cache.m7g.12xlarge | 6.2 | Yes | Yes | Yes | 22.5 | N/A | 157.12 |
| cache.m7g.16xlarge | 6.2 | Yes | Yes | Yes | 30 | N/A | 209.55 |
| cache.m6g.large | 5.0.6 | No | No | No | 0.75 | 10.0 | 6.38 |
| cache.m6g.xlarge | 5.0.6 | Yes | Yes | Yes | 1.25 | 10.0 | 12.93 |
| cache.m6g.2xlarge | 5.0.6 | Yes | Yes | Yes | 2.5 | 10.0 | 26.04 |
| cache.m6g.4xlarge | 5.0.6 | Yes | Yes | Yes | 5.0 | 10.0 | 52.26 |
| cache.m6g.8xlarge | 5.0.6 | Yes | Yes | Yes | 12 | N/A | 103.68 |
| cache.m6g.12xlarge | 5.0.6 | Yes | Yes | Yes | 20 | N/A | 157.12 |

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m6g.16xlarge | 5.0.6 | Yes | Yes | Yes | 25 | N/A | 209.55 |
| cache.m5.large | 4.0.10 | N | N | N | 0.75 | 10.0 | 6.38 |
| cache.m5.xlarge | 4.0.10 | Y | N | N | 1.25 | 10.0 | 12.93 |
| cache.m5.2xlarge | 4.0.10 | Y | Y | Y | 2.5 | 10.0 | 26.04 |
| cache.m5.4xlarge | 4.0.10 | Y | Y | Y | 5.0 | 10.0 | 52.26 |
| cache.m5.12xlarge | 4.0.10 | Y | Y | Y | 12 | N/A | 157.12 |
| cache.m5.24xlarge | 4.0.10 | Y | Y | Y | 25 | N/A | 314.32 |
| cache.m4.large | 4.0.10 | N | N | N | 0.45 | 1.2 | 6.42 |
| cache.m4.xlarge | 4.0.10 | Y | N | N | 0.75 | 2.8 | 14.28 |
| cache.m4.2xlarge | 4.0.10 | Y | Y | Y | 1.0 | 10.0 | 29.70 |

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.m4.4xlarge | 4.0.10 | Y | Y | Y | 2.0 | 10.0 | 60.78 |
| cache.m4.10xlarge | 4.0.10 | Y | Y | Y | 5.0 | 10.0 | 154.64 |
| cache.t4g.micro | 5.0.6 | No | No | No | 0.064 | 5.0 | 0.50 |
| cache.t4g.small | 5.0.6 | No | No | No | 0.128 | 5.0 | 1.37 |
| cache.t4g.medium | 5.0.6 | No | No | No | 0.256 | 5.0 | 3.09 |
| cache.t3.micro | 4.0.10 | N | N | N | 0.064 | 5.0 | 0.50 |
| cache.t3.small | 4.0.10 | N | N | N | 0.128 | 5.0 | 1.37 |
| cache.t3.medium | 4.0.10 | N | N | N | 0.256 | 5.0 | 3.09 |

### Memory optimized

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.r7g.large | 6.2 | Yes | No | Yes | 0.937 | 12.5 | 13.07 |
| cache.r7g.xlarge | 6.2 | Yes | Yes | Yes | 1.876 | 12.5 | 26.32 |
| cache.r7g.2xlarge | 6.2 | Yes | Yes | Yes | 3.75 | 15 | 52.82 |
| cache.r7g.4xlarge | 6.2 | Yes | Yes | Yes | 7.5 | 15 | 105.81 |
| cache.r7g.8xlarge | 6.2 | Yes | Yes | Yes | 15 | N/A | 209.55 |
| cache.r7g.12xlarge | 6.2 | Yes | Yes | Yes | 22.5 | N/A | 317.77 |
| cache.r7g.16xlarge | 6.2 | Yes | Yes | Yes | 30 | N/A | 419.09 |
| cache.r6g.large | 5.0.6 | No | No | No | 0.75 | 10.0 | 13.07 |
| cache.r6g.xlarge | 5.0.6 | Yes | Yes | Yes | 1.25 | 10.0 | 26.32 |
| cache.r6g.2xlarge | 5.0.6 | Yes | Yes | Yes | 2.5 | 10.0 | 52.82 |

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.r6g.4xlarge | 5.0.6 | Yes | Yes | Yes | 5.0 | 10.0 | 105.81 |
| cache.r6g.8xlarge | 5.0.6 | Yes | Yes | Yes | 12 | N/A | 209.55 |
| cache.r6g.12xlarge | 5.0.6 | Yes | Yes | Yes | 20 | N/A | 317.77 |
| cache.r6g.16xlarge | 5.0.6 | Yes | Yes | Yes | 25 | N/A | 419.09 |
| cache.r5.large | 4.0.10 | N | N | N | 0.75 | 10.0 | 13.07 |
| cache.r5.xlarge | 4.0.10 | Y | N | N | 1.25 | 10.0 | 26.32 |
| cache.r5.2xlarge | 4.0.10 | Y | Y | Y | 2.5 | 10.0 | 52.82 |
| cache.r5.4xlarge | 4.0.10 | Y | Y | Y | 5.0 | 10.0 | 105.81 |
| cache.r5.12xlarge | 4.0.10 | Y | Y | Y | 12 | N/A | 317.77 |
| cache.r5.24xlarge | 4.0.10 | Y | Y | Y | 25 | N/A | 635.61 |

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.r4.large | 4.0.10 | N | N | N | 0.75 | 10.0 | 12.30 |
| cache.r4.xlarge | 4.0.10 | Y | N | N | 1.25 | 10.0 | 25.05 |
| cache.r4.2xlarge | 4.0.10 | Y | Y | Y | 2.5 | 10.0 | 50.47 |
| cache.r4.4xlarge | 4.0.10 | Y | Y | Y | 5.0 | 10.0 | 101.38 |
| cache.r4.8xlarge | 4.0.10 | Y | Y | Y | 12 | N/A | 203.26 |
| cache.r4.16xlarge | 4.0.10 | Y | Y | Y | 25 | N/A | 407.00 |

### Memory optimized with data tiering

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) | SSD (GiB) |
|---|---|---|---|---|---|---|---|---|
| cache.r6gd.xlarge | 6.2.0 | Yes | No | No | 1.25 | 10 | 26.32 | 99.33 |
| cache.r6gd.2xlarge | 6.2.0 | Yes | Yes | Yes | 2.5 | 10 | 52.82 | 199.07 |
| cache.r6gd.4xlarge | 6.2.0 | Yes | Yes | Yes | 5.0 | 10 | 105.81 | 398.14 |
| cache.r6gd.8xlarge | 6.2.0 | Yes | Yes | Yes | 12 | N/A | 209.55 | 796.28 |
| cache.r6gd.12xlarge | 6.2.0 | Yes | Yes | Yes | 20 | N/A | 317.77 | 1194.42 |
| cache.r6gd.16xlarge | 6.2.0 | Yes | Yes | Yes | 25 | N/A | 419.09 | 1592.56 |

### Network optimized

| Instance type | Minimum supported Redis OSS version | Enhanced I/O with Redis OSS 5.0.6+ | TLS Offloading with Redis OSS 6.2.5+ | Enhanced I/O Multiplexing with Redis OSS 7.0.4+ | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|---|---|---|---|
| cache.c7gn.large | 6.2 | No | No | No | 6.25 | 30 | 3.09 |
| cache.c7gn.xlarge | 6.2 | Yes | Yes | Yes | 12.5 | 40 | 6.38 |
| cache.c7gn.2xlarge | 6.2 | Yes | Yes | Yes | 25 | 50 | 12.94 |
| cache.c7gn.4xlarge | 6.2 | Yes | Yes | Yes | 50 | N/A | 26.05 |
| cache.c7gn.8xlarge | 6.2 | Yes | Yes | Yes | 100 | N/A | 52.26 |
| cache.c7gn.12xlarge | 6.2 | Yes | Yes | Yes | 150 | N/A | 78.56 |
| cache.c7gn.16xlarge | 6.2 | Yes | Yes | Yes | 200 | N/A | 105.81 |

## Burstable Performance Nodes

You can launch general-purpose burstable T4g and T3-Standard cache nodes in Amazon ElastiCache. These nodes provide a baseline level of CPU performance with the ability to burst CPU usage at any time until the accrued credits are exhausted. A *CPU credit* provides the performance of a full CPU core for one minute.

Amazon ElastiCache's T4g and T3 nodes are configured as standard and suited for workloads with an average CPU utilization that is consistently below the baseline performance of the instance. To burst above the baseline, the node spends credits that it has accrued in its CPU credit balance. If the node is running low on accrued credits, performance is gradually lowered to the baseline performance level. This gradual lowering ensures the node doesn't experience a sharp performance drop-off when its accrued CPU credit balance is depleted. For more information, see CPU Credits and Baseline Performance for Burstable Performance Instances in the *Amazon EC2 User Guide*.

The following table lists the burstable performance node types, the rate at which CPU credits are earned per hour. It also shows the maximum number of earned CPU credits that a node can accrue and the number of vCPUs per node. In addition, it gives the baseline performance level as a percentage of a full core performance (using a single vCPU).

| CPU credits earned per hour | Maximum earned credits that can be accrued* | vCPUs | Baseline performance per vCPU | Network performance | Memory (GiB) |
|---|---|---|---|---|---|
| **Current generation** | | | | | |
| 12 | 288 | 2 | 10% | Up to 5 Gigabit | 0.5 |
| 24 | 576 | 2 | 20% | Up to 5 Gigabit | 1.37 |
| 24 | 576 | 2 | 20% | Up to 5 Gigabit | 3.09 |
| 12 | 288 | 2 | 10% | Up to 5 Gigabit | 0.5 |
| 24 | 576 | 2 | 20% | Up to 5 Gigabit | 1.37 |
| 24 | 576 | 2 | 20% | Up to 5 Gigabit | 3.09 |

| CPU credits earned per hour | Maximum earned credits that can be accrued* | vCPUs | Baseline performance per vCPU | Network performance | Memory (GiB) |
|---|---|---|---|---|---|
| **Previous generation** | | | | | |
| 6 | 144 | 1 | 10% | Low to moderate | 0.5 |
| 12 | 288 | 1 | 20% | Low to moderate | 1.55 |
| 24 | 576 | 2 | 20% | Low to moderate | 3.22 |

* The number of credits that can be accrued is equivalent to the number of credits that can be earned in a 24-hour period.

** The baseline performance in the table is per vCPU. Some node sizes that have more than one vCPU. For these, calculate the baseline CPU utilization for the node by multiplying the vCPU percentage by the number of vCPUs.

The following CPU credit metrics are available for T3 and T4g burstable performance instances:

- `CPUCreditUsage`
- `CPUCreditBalance`

For more information on these metrics, see CPU Credit Metrics.

In addition, be aware of these details:

- All current generation node types are created in a virtual private cloud (VPC) based on Amazon VPC by default.
- Redis OSS configuration variables `appendonly` and `appendfsync` aren't supported.

### Previous Generation

Previous generation node types are no longer available for new cluster creation. Existing clusters using these node types continue to function, but we recommend upgrading to current generation node types for better performance and cost efficiency.

| Instance type | Baseline bandwidth (Gbps) | Burst bandwidth (Gbps) | Memory (GiB) |
|---|---|---|---|
| cache.t2.micro | 0.064 | 1.024 | 0.555 |
| cache.t2.small | 0.128 | 1.024 | 1.55 |
| cache.t2.medium | 0.256 | 1.024 | 3.22 |

### Supported node types by AWS Region

Supported node types may vary between AWS Regions. For more details, see Amazon ElastiCache pricing.

### Related Information

- Amazon ElastiCache Product Features and Details
- Memcached Node-Type Specific Parameters for Memcached
- Valkey and Redis OSS parameters
- In Transit Encryption (TLS)

## Rebooting nodes

Some changes require that a Valkey, Memcached, or Redis OSS cluster reboot for the changes to be applied. For example, for some parameters, changing the parameter value in a parameter group is only applied after a reboot.

#### Topics

- Rebooting Valkey or Redis OSS nodes (cluster mode disabled only)
- Rebooting a cluster for Memcached

## Rebooting Valkey or Redis OSS nodes (cluster mode disabled only)

For Valkey or Redis OSS (cluster mode disabled) clusters, you can set the following parameters only when you create the cluster. After you associate a parameter group with a cluster, ElastiCache locks these parameters. To change them, you must create a new cluster with a parameter group that has the desired values:

- activerehashing
- databases

Valkey and Redis OSS nodes can only be updated through the ElastiCache console. You can only reboot a single node at a time. To reboot multiple nodes, you must repeat the process for each node.

**Valkey or Redis OSS (Cluster Mode Enabled) parameter changes**

If you make changes to the following parameters on a Valkey or Redis OSS (cluster mode enabled) cluster, follow the ensuing steps.

- activerehashing
- databases

1. Create a manual backup of your cluster. See Taking manual backups.
2. Delete the Valkey or Redis OSS (cluster mode enabled) cluster. See Deleting a cluster in ElastiCache.
3. Restore the cluster using the altered parameter group and backup to seed the new cluster. See Restoring from a backup into a new cache.

Changes to other parameters do not require this.

### Using the AWS Management Console

You can reboot a node using the ElastiCache console.

**To reboot a node (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region that applies.
3. In the left navigation pane, choose **Valkey or Redis OSS**.

   A list of clusters running Valkey or Redis OSS appears.
4. Choose the cluster under **Cluster Name**.
5. Under **Node name**, choose the radio button next to the node you want to reboot.
6. Choose **Actions**, and then choose **Reboot node**.

To reboot multiple nodes, repeat steps 2 through 5 for each node that you want to reboot. You do not need to wait for one node to finish rebooting to reboot another.

## Rebooting a cluster for Memcached

When you reboot a Memcached cluster, the cluster flushes all its data and restarts its engine. During this process you cannot access the cluster. Because the cluster flushed all its data, when the cluster is available again, you are starting with an empty cluster.

You are able to reboot a cluster using the ElastiCache console, the AWS CLI, or the ElastiCache API. Whether you use the ElastiCache console, the AWS CLI or the ElastiCache API, you can only initiate rebooting a single cluster. To reboot multiple clusters you must iterate on the process or operation.

**Using the AWS Management Console**

You can reboot a cluster using the ElastiCache console.

**To reboot a cluster (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region you are interested in.
3. In the navigation pane, choose the engine running on the cluster that you want to reboot.

   A list of clusters running the chosen engine appears.

4. Choose the cluster to reboot by choosing on the box to the left of the cluster's name.

   The **Reboot** button becomes active.

   If you choose more than one cluster, the **Reboot** button isn't active.

5. Choose **Reboot**.

   The reboot cluster confirmation screen appears.

6. To reboot the cluster, choose **Reboot**. The status of the cluster changes to *rebooting cluster nodes*.

   To not reboot the cluster, choose **Cancel**.

To reboot multiple clusters, repeat steps 2 through 5 for each cluster that you want to reboot. You do not need to wait for one cluster to finish rebooting to reboot another.

To reboot a specific node, select the node and then choose **Reboot**.

**Using the AWS CLI**

To reboot a cluster (AWS CLI), use the `reboot-cache-cluster` CLI operation.

To reboot specific nodes in the cluster, use the `--cache-node-ids-to-reboot` to list the specific clusters to reboot. The following command reboots the nodes 0001, 0002, and 0004 of *my-cluster*.

For Linux, macOS, or Unix:

```
aws elasticache reboot-cache-cluster \
    --cache-cluster-id my-cluster \
    --cache-node-ids-to-reboot 0001 0002 0004
```

For Windows:

```
aws elasticache reboot-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --cache-node-ids-to-reboot 0001 0002 0004
```

To reboot all the nodes in the cluster, use the `--cache-node-ids-to-reboot` parameter and list all the cluster's node ids. For more information, see reboot-cache-cluster.

### Using the ElastiCache API

To reboot a cluster using the ElastiCache API, use the `RebootCacheCluster` action.

To reboot specific nodes in the cluster, use the `CacheNodeIdsToReboot` to list the specific clusters to reboot. The following command reboots the nodes 0001, 0002, and 0004 of *my-cluster*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=RebootCacheCluster
   &CacheClusterId=my-cluster
   &CacheNodeIdsToReboot.member.1=0001
   &CacheNodeIdsToReboot.member.2=0002
   &CacheNodeIdsToReboot.member.3=0004
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

To reboot all the nodes in the cluster, use the `CacheNodeIdsToReboot` parameter and list all the cluster's node ids. For more information, see RebootCacheCluster.

# Replacing nodes (Valkey and Redis OSS)

Amazon ElastiCache frequently upgrades its fleet with patches and upgrades being applied to instances seamlessly. However, from time to time we need to relaunch your ElastiCache nodes to apply mandatory OS updates to the underlying host. These replacements are required to apply upgrades that strengthen security, reliability, and operational performance.

You have the option to manage these replacements yourself at any time before the scheduled node replacement window. When you manage a replacement yourself, your instance receives the OS update when you relaunch the node and your scheduled node replacement is canceled. You might continue to receive alerts indicating that the node replacement is to take place. If you've already manually mitigated the need for the maintenance, you can ignore these alerts.

> **Note**
>
> Replacement cache nodes automatically generated by Amazon ElastiCache may have different IP addresses. You are responsible for reviewing your application configuration to ensure that your cache nodes are associated with the appropriate IP addresses.

The following list identifies actions you can take when ElastiCache schedules one of your Valkey or Redis OSS nodes for replacement. To expedite finding the information you need for your situation, choose from the following menu.

- Do nothing – Let Amazon ElastiCache replace the node as scheduled.
- Change your maintenance window – Change your maintenance window to a better time.
- Valkey or Redis OSS (cluster mode enabled) Configurations
  - Replace the only node in any Valkey or Redis OSS cluster – A procedure to replace a node in a Valkey or Redis OSS cluster using backup and restore.
  - Replace a replica node in any Valkey or Redis OSS cluster – A procedure to replace a read-replica in any Valkey or Redis OSS cluster by increasing and decreasing the replica count with no cluster downtime.
  - Replace any node in a Valkey or Redis OSS (cluster mode enabled) shard – A dynamic procedure with no cluster downtime to replace a node in a Valkey or Redis OSS (cluster mode enabled) cluster by scaling out and scaling in.
- Valkey or Redis OSS (cluster mode disabled) Configurations

- Replace the only node in any Valkey or Redis OSS cluster – Procedure to replace any node in a Valkey or Redis OSS cluster using backup and restore.
- Replace a replica node in any Valkey or Redis OSS cluster – A procedure to replace a read-replica in any Valkey or Redis OSS cluster by increasing and decreasing the replica count with no cluster downtime.
- Replace a node in a Valkey or Redis OSS (cluster mode disabled) cluster – Procedure to replace a node in a Valkey or Redis OSS (cluster mode disabled) cluster using replication.
- Replace a Valkey or Redis OSS (cluster mode disabled) read-replica – A procedure to manually replace a read-replica in a Valkey or Redis OSS (cluster mode disabled) replication group.
- Replace a Valkey or Redis OSS (cluster mode disabled) primary node – A procedure to manually replace the primary node in a Valkey or Redis OSS (cluster mode disabled) replication group.

**Valkey and Redis OSS node replacement options**

- **Do nothing** – If you do nothing, ElastiCache replaces the node as scheduled.

  For non-Cluster configurations with autofailover enabled, clusters on Valkey 7.2 and above and Redis OSS 5.0.6 and above complete replacement while the cluster continues to stay online and serve incoming write requests. For auto failover enabled clusters on Redis OSS 4.0.10 or below, you might notice a brief write interruption of up to a few seconds associated with DNS updates.

  If the node is a member of an auto failover enabled cluster, ElastiCache for Valkey or Redis OSS provides improved availability during patching, updates, and other maintenance-related node replacements.

  For ElastiCache cluster configurations that are set up to use ElastiCache for Valkey or Redis OSS cluster clients, replacement now completes while the cluster serves incoming write requests.

  For non-cluster configurations with autofailover enabled, clusters on Valkey 7.2 and above and Redis OSS 5.0.6 and above complete replacement while the cluster continues to stay online and serve incoming write requests. For auto failover enabled clusters on Redis OSS 4.0.10 or below, you might notice a brief write interruption of up to a few seconds associated with DNS updates.

If the node is standalone, Amazon ElastiCache first launches a replacement node and then syncs from the existing node. The existing node isn't available for service requests during this time. Once the sync is complete, the existing node is terminated and the new node takes its place. ElastiCache makes a best effort to retain your data during this operation.

- **Change your maintenance window** – For scheduled maintenance events, you receive an email or a notification event from ElastiCache. In these cases, if you change your maintenance window before the scheduled replacement time, your node now is replaced at the new time. For more information, see the following:
  - Modifying an ElastiCache cluster
  - Modifying a replication group

  > **Note**
  >
  > The ability to change your replacement window by moving your maintenance window is only available when the ElastiCache notification includes a maintenance window. If the notification does not include a maintenance window, you cannot change your replacement window.

  For example, let's say it's Thursday, November 9, at 15:00 and the next maintenance window is Friday, November 10, at 17:00. Following are three scenarios with their outcomes:

  - You change your maintenance window to Fridays at 16:00, after the current date and time and before the next scheduled maintenance window. The node is replaced on Friday, November 10, at 16:00.
  - You change your maintenance window to Saturday at 16:00, after the current date and time and after the next scheduled maintenance window. The node is replaced on Saturday, November 11, at 16:00.
  - You change your maintenance window to Wednesday at 16:00, earlier in the week than the current date and time). The node is replaced next Wednesday, November 15, at 16:00.

  For instructions, see Managing ElastiCache cluster maintenance.

- **Replace the only node in any Valkey or Redis OSS cluster** – If the cluster does not have any read replicas, you can use the following procedure to replace the node.

  **To replace the only node using backup and restore**

  1. Create a snapshot of the node's cluster. For instructions, see Taking manual backups.
  2. Create a new cluster seeding it from the snapshot. For instructions, see Restoring from a backup into a new cache.
  3. Delete the cluster with the node scheduled for replacement. For instructions, see Deleting a cluster in ElastiCache.
  4. In your application, replace the old node's endpoint with the new node's endpoint.

- **Replace a replica node in any Valkey or Redis OSS cluster** – To replace a replica cluster, increase your replica count. To do this, add a replica then decrease the replica count by removing the replica that you want to replace. This process is dynamic and doesn't have any cluster downtime.

  > **Note**
  >
  > If your shard or replication group already has five replicas, reverse steps 1 and 2.

  **To replace a replica in any Valkey or Redis OSS cluster**

  1. Increase the replica count by adding a replica to the shard or replication group. For more information, see Increasing the number of replicas in a shard.
  2. Delete the replica you want to replace. For more information, see Decreasing the number of replicas in a shard.
  3. Update the endpoints in your application.

- **Replace any node in a Valkey or Redis OSS (cluster mode enabled) shard** – To replace the node in a cluster with no downtime, use online resharding. First add a shard by scaling out, and then delete the shard with the node to be replaced by scaling in.

**To replace any node in a Valkey or Redis OSS (cluster mode enabled) cluster**

1. Scale out: Add an additional shard with the same configuration as the existing shard with the node to be replaced. For more information, see Adding shards with online resharding.
2. Scale in: Delete the shard with the node to be replaced. For more information, see Removing shards with online resharding.
3. Update the endpoints in your application.

- **Replace a node in a Valkey or Redis OSS (cluster mode disabled) cluster** – If the cluster is a Valkey or Redis OSS (cluster mode disabled) cluster without any read replicas, use the following procedure to replace the node.

  **To replace the node using replication (cluster mode disabled only)**

  1. Add replication to the cluster with the node scheduled for replacement as the primary. Do not enable Multi-AZ on this cluster. For instructions, see To add replication to a Valkey or Redis OSS cluster with no shards.
  2. Add a read-replica to the cluster. For instructions, see To add nodes to an ElastiCache cluster (console).
  3. Promote the newly created read-replica to primary. For instructions, see Promoting a read replica to primary, for Valkey or Redis OSS (cluster mode disabled) replication groups.
  4. Delete the node scheduled for replacement. For instructions, see Removing nodes from an ElastiCache cluster.
  5. In your application, replace the old node's endpoint with the new node's endpoint.

- **Replace a Valkey or Redis OSS (cluster mode disabled) read-replica** – If the node is a read-replica, replace the node.

  If your cluster has only one replica node and Multi-AZ is enabled, you must disable Multi-AZ before you can delete the replica. For instructions, see Modifying a replication group.

  **To replace a Valkey or Redis OSS (cluster mode disabled) read replica**

  1. Delete the replica that is scheduled for replacement. For instructions, see the following:

- Decreasing the number of replicas in a shard
- Removing nodes from an ElastiCache cluster

2. Add a new replica to replace the one that is scheduled for replacement. If you use the same name as the replica you just deleted, you can skip step 3. For instructions, see the following:
   - Increasing the number of replicas in a shard
   - Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled)
3. In your application, replace the old replica's endpoint with the new replica's endpoint.
4. If you disabled Multi-AZ at the start, re-enable it now. For instructions, see Enabling Multi-AZ.

- **Replace a Valkey or Redis OSS (cluster mode disabled) primary node** – If the node is the primary node, first promote a read-replica to primary. Then delete the replica that used to be the primary node.

  If your cluster has only one replica and Multi-AZ is enabled, you must disable Multi-AZ before you can delete the replica in step 2. For instructions, see Modifying a replication group.

  **To replace a Valkey or Redis OSS (cluster mode disabled) primary node**

  1. Promote a read-replica to primary. For instructions, see Promoting a read replica to primary, for Valkey or Redis OSS (cluster mode disabled) replication groups.
  2. Delete the node that is scheduled for replacement (the old primary). For instructions, see Removing nodes from an ElastiCache cluster.
  3. Add a new replica to replace the one scheduled for replacement. If you use the same name as the node you just deleted, you can skip changing endpoints in your application.

     For instructions, see Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled).
  4. In your application, replace the old node's endpoint with the new node's endpoint.
  5. If you disabled Multi-AZ at the start, re-enable it now. For instructions, see Enabling Multi-AZ.

# Replacing nodes (Memcached)

Amazon ElastiCache for Memcached frequently upgrades its fleet with patches and upgrades being applied to instances seamlessly. However, from time to time we need to relaunch your ElastiCache for Memcached nodes to apply mandatory OS updates to the underlying host. These replacements are required to apply upgrades that strengthen security, reliability, and operational performance.

You have the option to manage these replacements yourself at any time before the scheduled node replacement window. When you manage a replacement yourself, your instance receives the OS update when you relaunch the node and your scheduled node replacement is canceled. You might continue to receive alerts indicating that the node replacement takes place. If you already manually mitigated the need for the maintenance, you can ignore these alerts.

> **Note**
>
> Replacement cache nodes automatically generated by Amazon ElastiCache may have different IP addresses. You are responsible for reviewing your application configuration to ensure that your cache nodes are associated with the appropriate IP addresses.

The following list identifies actions you can take when ElastiCache schedules one of your Memcached nodes for replacement.

- **Do nothing** – If you do nothing, ElastiCache replaces the node as scheduled. When ElastiCache automatically replaces the node with a new node, the new node is initially empty.
- **Change your maintenance window** – For scheduled maintenance events, you receive an email or a notification event from ElastiCache. In this case, if you change your maintenance window before the scheduled replacement time, your node now is replaced at the new time. For more information, see Modifying an ElastiCache cluster.

  > **Note**
  >
  > The ability to change your replacement window by moving your maintenance window is only available when the ElastiCache notification includes a maintenance window. If the notification does not include a maintenance window, you cannot change your replacement window.

For example, let's say it's Thursday, November 9, at 15:00 and the next maintenance window is Friday, November 10, at 17:00. Following are three scenarios with their outcomes:

- You change your maintenance window to Fridays at 16:00, after the current date and time and before the next scheduled maintenance window. The node is replaced on Friday, November 10, at 16:00.
- You change your maintenance window to Saturday at 16:00, after the current date and time and after the next scheduled maintenance window. The node is replaced on Saturday, November 11, at 16:00.
- You change your maintenance window to Wednesday at 16:00, earlier in the week than the current date and time). The node is replaced next Wednesday, November 15, at 16:00.

For instructions, see Managing ElastiCache cluster maintenance.

- **Manually replace the node** – If you need to replace the node before the next maintenance window, manually replace the node.

  If you manually replace the node, keys are redistributed. This redistribution causes cache misses.

  **To manually replace a Memcached node**

  1. Delete the node scheduled for replacement. For instructions, see Removing nodes from an ElastiCache cluster.
  2. Add a new node to the cluster. For instructions, see Adding nodes to an ElastiCache cluster.
  3. If you are not using auto discovery on this cluster, see your application and replace every instance of the old node's endpoint with the new node's endpoint.

## Reserved nodes

**Topics**


- Managing costs with reserved nodes
- Standard reserved node offerings
- Size flexible reserved nodes
- Deleting a reserved node
- Legacy reserved node offerings
- Getting info about reserved node offerings

- Purchasing a reserved node
- Getting info about your reserved nodes

## Managing costs with reserved nodes

Reserving one or more nodes may be a way for you to reduce costs. Reserved nodes offer discounted hourly rates compared to On-Demand pricing, with options for No Upfront, Partial Upfront, or All Upfront payment for one or three-year terms.

To see if reserved nodes are a cost savings for your use cases, first determine the node size and number of nodes you need. Then estimate the usage of the node, and compare the total cost to you using On-Demand nodes versus reserved nodes. You can mix and match reserved and On-Demand node usage in your clusters. For pricing information, see Amazon ElastiCache Pricing.

AWS Region, node type and term length must be chosen at purchase, and cannot be changed later.

You can use the AWS Management Console, the AWS CLI, or the ElastiCache API to list and purchase available reserved node offerings.

For more information on reserved nodes, see Amazon ElastiCache Reserved Nodes.

## Standard reserved node offerings

When you purchase a reserved node instance (RI) in Amazon ElastiCache, you can purchase a commitment to getting a discounted rate on a specific node instance type and AWS Region for the duration of the reserved node instance. To use an Amazon ElastiCache reserved node instance, you create a new ElastiCache node instance, just as you would for an on-demand instance.

If the specifications of the new reserve node instance match an existing reserved node instance for your account, you are billed at the discounted rate offered for the reserved node instance. Otherwise, the node instance is billed at an on-demand rate. These standard RIs are available from R5 and M5 instance families onwards.

> **Note**
>
> All offering types discussed next are available in one-year and three-year terms.

### Offering Types

**No Upfront** RI provides access to a reserved ElastiCache instance without requiring an upfront payment. Your *No Upfront* reserved ElastiCache instance bills a discounted hourly rate for every hour within the term, regardless of usage.

**Partial Upfront** RI requires a part of the reserved ElasticCache instance to be paid upfront. The remaining hours in the term are billed at a discounted hourly rate, regardless of usage. This option is the replacement for the legacy *Heavy Utilization* option, which is explained in the next section.

**All Upfront** RI requires full payment to be made at the start of the RI term. You incur no other costs for the remainder of the term, regardless of the number of hours used.

## Size flexible reserved nodes

All reserved nodes are size flexible. When you purchase a reserved node, one thing that you specify is the node type, for example cache.r6g.xlarge. For more information, about node types, see Amazon ElastiCache Pricing.

If you have a node, and you need to scale it to larger capacity, your reserved node is automatically applied to your scaled node. That is, your reserved nodes are automatically applied to usage of any size in the same node family. Size-flexible reserved nodes are available for nodes with the same AWS Region. Size-flexible reserved nodes can only scale in their node families. For example, a reserved node for a cache.r6g.xlarge can apply to a cache.r6g.2xlarge, but not to a cache.r6gd.large, because cache.r6g and cache.r6gd are different node families.

Size flexibility means that you can move freely between configurations within the same node family. For example, you can move from a r6g.xlarge reserved node (8 normalized units) to two r6g.large reserved nodes (8 normalized units) (2*4 = 8 normalized units) in the same AWS Region at no extra cost.

### Upgrading nodes from Redis OSS to Valkey

With the launch of Valkey in ElastiCache, you can now apply your Redis OSS reserved node discount to the Valkey cache engine. You can upgrade from Redis OSS to Valkey while still benefitting from existing contracts and reservations. In addition to being able to apply your benefits within the cache node family and engine, you can even receive more incremental value. Valkey is priced at a 20% discount relative to Redis OSS, and with reserved node flexibility, you can use your Redis OSS reserved nodes to cover 20% more running Valkey nodes.

To calculate the discounted rate, each ElastiCache node and engine combination has a normalization factor that's measured in units. Reserved node units can be applied to any running

node within the reserved node's instance family for a given engine. Redis OSS reserved nodes can additionally apply across engines to cover running Valkey nodes. Valkey reserved nodes apply only to running Valkey nodes within the same instance family. Because Valkey is priced at a discount relative to Redis OSS and Memcached, its units for a given instance type are lower, which allows a Redis OSS reserved node to cover more Valkey nodes.

As an example, let's say you have purchased a reserved node for a cache.r7g.4xlarge for the Redis OSS engine (32 units) and are running one cache.r7g.4xlarge Redis OSS node (32 units). If you upgrade the node to Valkey, the normalization factor of the running node drops to 25.6 units, and your existing reserved node provides you with an additional 6.4 units to use against any other running Valkey or Redis OSS node within the cache.r7g family in the Region. You could use this to cover 25% of another cache.r7g.4xlarge Valkey node in the account (25.6 units), or 100% of a cache.r7g.xlarge Valkey node (6.4 units).

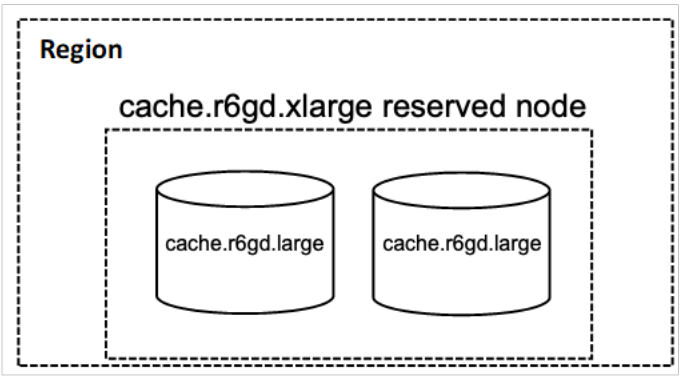
### Comparing usage with normalized units

You can compare usage for different reserved node sizes by using normalized units. For example, one hour of usage on two cache.r6g.4xlarge nodes is equivalent to 16 hours of usage on one cache.r6g.large. The following table shows the number of normalized units for each node size:

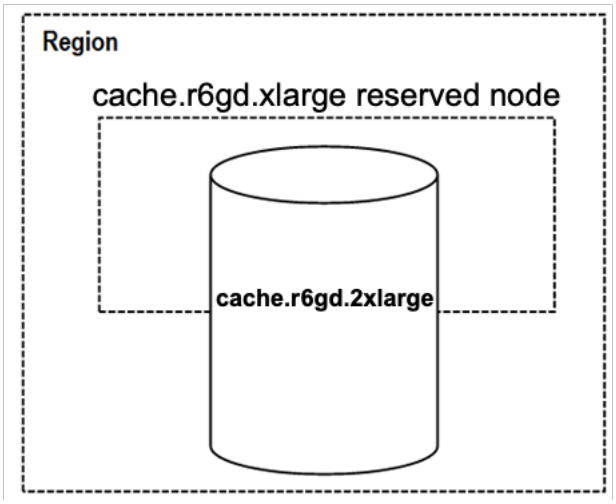
| Node size | Normalized units with Redis OSS or Memcached | Normalized units with Valkey |
|---|---|---|
| micro | 0.5 | 0.4 |
| small | 1 | .8 |
| medium | 2 | 1.6 |
| large | 4 | 3.2 |
| xlarge | 8 | 6.4 |
| 2xlarge | 16 | 12.8 |
| 4xlarge | 32 | 25.6 |
| 6xlarge | 48 | 38.4 |
| 8xlarge | 64 | 51.2 |

| Node size | Normalized units with Redis OSS or Memcached | Normalized units with Valkey |
|---|---|---|
| 10xlarge | 80 | 64 |
| 12xlarge | 96 | 76.8 |
| 16xlarge | 128 | 102.4 |
| 24xlarge | 192 | 153.6 |

For example, you purchase a cache.r6gd.xlarge reserved node, and you have two running cache.r6gd.large reserved nodes in your account in the same AWS Region. In this case, the billing benefit is applied in full to both nodes.




Alternatively, if you have one cache.r6gd.2xlarge instance running in your account in the same AWS Region, the billing benefit is applied to 50 percent of the usage of the reserved node.



## Deleting a reserved node

The terms for a reserved node involve a one-year or three-year commitment. You can't cancel a reserved node. However, you can delete a node that is covered by a reserved node discount. The process for deleting a node that is covered by a reserved node discount is the same as for any other node.

If you delete a node that is covered by a reserved node discount, you can launch another node with compatible specifications. In this case, you continue to get the discounted rate during the reservation term (one or three years).

## Legacy reserved node offerings

There are three levels of legacy node reservations—Heavy Utilization, Medium Utilization, and Light Utilization. Nodes can be reserved at any utilization level for either one or three years. The node type, utilization level, and reservation term affect your total costs. Verify the savings that reserved nodes can provide your business by comparing various models before you purchase reserved nodes.

Nodes purchased at one utilization level or term cannot be converted to a different utilization level or term.

### Utilization Levels

*Heavy Utilization reserved nodes* enable workloads that have a consistent baseline of capacity or run steady-state workloads. Heavy Utilization reserved nodes require a high up-front commitment, but if you plan to run more than 79 percent of the reserved node term you can earn the largest savings (up to 70 percent off of the On-Demand price). With Heavy Utilization reserved nodes, you pay a one-time fee. This is then followed by a lower hourly fee for the duration of the term regardless of whether your node is running.

*Medium Utilization reserved nodes* are the best option if you plan to use your reserved nodes a large amount of the time and you want either a lower one-time fee or to stop paying for your node when you shut it off. Medium Utilization reserved nodes are a more cost-effective option when you plan to run more than 40 percent of the reserved nodes term. This option can save you up to 64 percent off of the On-Demand price. With Medium Utilization reserved nodes, you pay a slightly higher one-time fee than with Light Utilization reserved nodes, and you receive lower hourly usage rates when you run a node.

*Light Utilization reserved nodes* are ideal for periodic workloads that run only a couple of hours a day or a few days per week. Using Light Utilization reserved nodes, you pay a one-time fee followed by a discounted hourly usage fee when your node is running. You can start saving when your node is running more than 17 percent of the reserved node term. You can save up to 56 percent off of the On-Demand rates over the entire term of your reserved node.

**Legacy reserved node offerings**

| Offering | Up-front cost | Usage fee | Advantage |
|---|---|---|---|
| Heavy Utilization | Highest | Lowest hourly fee. Applied to the whole term whether or not you're using the reserved node. | Lowest overall cost if you plan to run your reserved nodes more than 79 percent of a three-year term. |
| Medium Utilization | Medium | Hourly usage fee charged for each hour the node is running. No hourly charge when the node is not running. | Suitable for elastic workloads or when you expect moderate usage, more than 40 percent of a three-year term. |
| Light Utilization | Lowest | Hourly usage fee charged for each hour the node is running. No hourly charge when the node is not running. Highest hourly fees of all the offering types, but fees apply only when the reserved node is running. | Highest overall cost if you plan to run all of the time. However, this is the lowest overall cost if you plan to use your reserved node infrequently, more than about 15 percent of a three-year term. |

| Offering | Up-front cost | Usage fee | Advantage |
| --- | --- | --- | --- |
| On-Demand Use (No reserved nodes) | None | Highest hourly fee. Applied whenever the node is running. | Highest hourly cost. |

For more information, see Amazon ElastiCache Pricing.

## Getting info about reserved node offerings

Before you purchase reserved nodes, you can get information about available reserved node offerings.

The following examples show how to get pricing and information about available reserved node offerings using the AWS Management Console, AWS CLI, and ElastiCache API.

**Topics**


- Getting info about reserved node offerings (Console)
- Getting info about reserved node offerings (AWS CLI)
- Getting info about reserved node offerings (ElastiCache API)


### Getting info about reserved node offerings (Console)

To get pricing and other information about available reserved cluster offerings using the AWS Management Console, use the following procedure.

**To get information about available reserved node offerings**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Reserved Nodes**.
3. Choose **Purchase Reserved Node**.
4. For **Engine**, choose either Valkey, Memcached, or Redis OSS.
5. To determine the available offerings, make selections for the following options:

   - **Node Type**
   - **Term**
   - **Offering Type**

   After you make these selections, the cost per node and total cost of your selections is shown under **Reservation details**.
6. Choose **Cancel** to avoid purchasing these nodes and incurring charges.

### Getting info about reserved node offerings (AWS CLI)

To get pricing and other information about available reserved node offerings for Valkey or Redis OSS, type the following command at a command prompt:

```
aws elasticache describe-reserved-cache-nodes-offerings
```

This operation produces output similar to the following (JSON format):

```
{
          "ReservedCacheNodesOfferingId": "0xxxxxxxx-xxeb-44ex-xx3c-xxxxxxxx072",
          "CacheNodeType": "cache.xxx.large",
          "Duration": 94608000,
          "FixedPrice": XXXX.X,
          "UsagePrice": X.X,
          "ProductDescription": "redis",
          "OfferingType": "All Upfront",
          "RecurringCharges": [
              {
                  "RecurringChargeAmount": X.X,
                  "RecurringChargeFrequency": "Hourly"
              }
          ]
 },
 {
          "ReservedCacheNodesOfferingId": "0xxxxxxxx-xxeb-44ex-xx3c-xxxxxxxx072",
          "CacheNodeType": "cache.xxx.xlarge",
          "Duration": 94608000,
          "FixedPrice": XXXX.X,
          "UsagePrice": X.X,
          "ProductDescription": "redis",
          "OfferingType": "Partial Upfront",
          "RecurringCharges": [
              {
                  "RecurringChargeAmount": X.XXX,
                  "RecurringChargeFrequency": "Hourly"
              }
          ]
 },
 {
          "ReservedCacheNodesOfferingId": "0xxxxxxxx-xxeb-44ex-xx3c-xxxxxxxx072",
          "CacheNodeType": "cache.xxx.large",
          "Duration": 31536000,
```

```
            "FixedPrice": X.X,
            "UsagePrice": X.X,
            "ProductDescription": "redis",
            "OfferingType": "No Upfront",
            "RecurringCharges": [
                {
                    "RecurringChargeAmount": X.XXX,
                    "RecurringChargeFrequency": "Hourly"
                }
            ]
}
```

To get pricing and other information about available reserved node offerings for Memcached, type the following command at a command prompt:

```
 {
            "ReservedCacheNodesOfferingId": "0xxxxxxxx-xxeb-44ex-xx3c-xxxxxxxx072",
            "CacheNodeType": "cache.xxx.large",
            "Duration": 94608000,
            "FixedPrice": XXXX.X,
            "UsagePrice": X.X,
            "ProductDescription": "memcached",
            "OfferingType": "All Upfront",
            "RecurringCharges": [
                {
                    "RecurringChargeAmount": X.X,
                    "RecurringChargeFrequency": "Hourly"
                }
            ]
    },
 {
            "ReservedCacheNodesOfferingId": "0xxxxxxxx-xxeb-44ex-xx3c-xxxxxxxx072",
            "CacheNodeType": "cache.xxx.xlarge",
            "Duration": 94608000,
            "FixedPrice": XXXX.X,
            "UsagePrice": X.X,
            "ProductDescription": "memcached",
            "OfferingType": "Partial Upfront",
            "RecurringCharges": [
                {
                    "RecurringChargeAmount": X.XXXX,
                    "RecurringChargeFrequency": "Hourly"
                }
```

```
            ]
        },
        {
            "ReservedCacheNodesOfferingId": "0xxxxxxxx-xxeb-44ex-xx3c-xxxxxxxx072",
            "CacheNodeType": "cache.xx.12xlarge",
            "Duration": 31536000,
            "FixedPrice": X.X,
            "UsagePrice": X.X,
            "ProductDescription": "memcached",
            "OfferingType": "No Upfront",
            "RecurringCharges": [
                {
                    "RecurringChargeAmount": X.XXXX,
                    "RecurringChargeFrequency": "Hourly"
                }
            ]
}
```

For more information, see describe-reserved-cache-nodes-offerings in the AWS CLI Reference.

**Getting info about reserved node offerings (ElastiCache API)**

To get pricing and information about available reserved node offerings, call the `DescribeReservedCacheNodesOfferings` action.

**Example**

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeReservedCacheNodesOfferings
    &Version=2014-12-01
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see DescribeReservedCacheNodesOfferings in the ElastiCache API Reference.

## Purchasing a reserved node

The following examples show how to purchase a reserved node offering using the AWS Management Console, the AWS CLI, and the ElastiCache API.

> **Important**
>
> Following the examples in this section incurs charges on your AWS account that you can't reverse.

**Topics**


- Purchasing a reserved node (Console)
- Purchasing a reserved node (AWS CLI)
- Purchasing a reserved node (ElastiCache API)


### Purchasing a reserved node (Console)

This example shows purchasing a specific reserved node offering, *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*, with a reserved node ID of *myreservationID*.

The following procedure uses the AWS Management Console to purchase the reserved node offering by offering id.

**To purchase reserved nodes**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation list, choose the **Reserved Nodes** link.
3. Choose the **Purchase reserved nodes** button.
4. For **Engine**, choose Valkey, Memcached, or Redis OSS.
5. To determine the available offerings, make selections for the following options:
   - **Node Type**
   - **Term**
   - **Offering Type**
   - An optional **Reserved node ID**

After you make these selections, the cost per node and total cost of your selections is shown under **Reservation details**.

6. Choose **Purchase**.

### Purchasing a reserved node (AWS CLI)

The following example shows purchasing the specific reserved cluster offering, *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*, with a reserved node ID of *myreservationID*.

Type the following command at a command prompt:

For Linux, macOS, or Unix:

```
aws elasticache purchase-reserved-cache-nodes-offering \
    --reserved-cache-nodes-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-cache-node-id myreservationID
```

For Windows:

```
aws elasticache purchase-reserved-cache-nodes-offering ^
    --reserved-cache-nodes-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-cache-node-id myreservationID
```

The command returns output similar to the following:

```
RESERVATION  ReservationId      Class           Start Time                Duration
 Fixed Price  Usage Price  Count  State            Description      Offering Type
RESERVATION  myreservationid    cache.xx.small  2013-12-19T00:30:23.247Z  1y
 XXX.XX USD   X.XXX USD    1      payment-pending  memcached        Medium Utilization
```

For more information, see purchase-reserved-cache-nodes-offering in the AWS CLI Reference.

### Purchasing a reserved node (ElastiCache API)

The following example shows purchasing the specific reserved node offering, *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*, with a reserved cluster ID of *myreservationID*.

Call the `PurchaseReservedCacheNodesOffering` operation with the following parameters:

- ReservedCacheNodesOfferingId = 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f
- ReservedCacheNodeID = myreservationID
- CacheNodeCount = 1

### Example

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=PurchaseReservedCacheNodesOffering
    &ReservedCacheNodesOfferingId=649fd0c8-cf6d-47a0-bfa6-060f8e75e95f
    &ReservedCacheNodeID=myreservationID
    &CacheNodeCount=1
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see PurchaseReservedCacheNodesOffering in the ElastiCache API Reference.

## Getting info about your reserved nodes

You can get information about the reserved nodes you've purchased using the AWS Management Console, the AWS CLI, and the ElastiCache API.

**Topics**


- Getting info about your reserved nodes (Console)
- Getting info about your reserved nodes (AWS CLI)
- Getting info about your reserved nodes (ElastiCache API)


### Getting info about your reserved nodes (Console)

The following procedure describes how to use the AWS Management Console to get information about the reserved nodes you purchased.

**To get information about your purchased reserved nodes**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation list, choose the **Reserved nodes** link.

   The reserved nodes for your account appear in the Reserved nodes list. You can choose any of the reserved nodes in the list to see detailed information about the reserved node in the detail pane at the bottom of the console.

### Getting info about your reserved nodes (AWS CLI)

To get information about reserved nodes for your AWS account, type the following command at a command prompt:

```
aws elasticache describe-reserved-cache-nodes
```

This operation produces output similar to the following (JSON format):

```
{
    "ReservedCacheNodeId": "myreservationid",
    "ReservedCacheNodesOfferingId": "649fd0c8-cf6d-47a0-bfa6-060f8e75e95f",
    "CacheNodeType": "cache.xx.small",
    "DataTiering": "disabled",
```

```
    "Duration": "31536000",
    "ProductDescription": "memcached",
    "OfferingType": "Medium Utilization",
    "MaxRecords": 0
}
```

For more information, see describe-reserved-cache-nodes in the AWS CLI Reference.

**Getting info about your reserved nodes (ElastiCache API)**

To get information about reserved nodes for your AWS account, call the `DescribeReservedCacheNodes` operation.

**Example**

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeReservedCacheNodes
    &Version=2014-12-01
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see DescribeReservedCacheNodes in the ElastiCache API Reference.

## Migrating previous generation nodes

Previous generation nodes are node types that are being phased out. If you have no existing clusters using a previous generation node type, ElastiCache does not support the creation of new clusters with that node type.

Due to the limited amount of previous generation node types, we cannot guarantee a successful replacement when a node becomes unhealthy in your cluster(s). In such a scenario, your cluster availability may be negatively impacted.

We recommend that you migrate your cluster(s) to a new node type for better availability and performance. For a recommended node type to migrate to, see Upgrade Paths. For a full list of

supported node types and previous generation node types in ElastiCache, see Supported node types.

## Migrating nodes on a Valkey or Redis OSS cluster

The following procedure describes how to migrate your Valkey or Redis OSS cluster node type using the ElastiCache Console. During this process, your Valkey or Redis OSS cluster will continue to serve requests with minimal downtime. Depending on your cluster configuration you may see the following downtimes. The following are estimates and may differ based on your specific configurations:

- Cluster mode disabled (single node) may see approximately 60 seconds, primarily due to DNS propagation.
- Cluster mode disabled (with replica node) may see approximately 1 second for clusters running Valkey 7.2 and above or Redis OSS 5.0.6 and above. All lower versions can experience approximately 10 seconds.
- Cluster mode enabled may see approximately 1 second.

**To modify a Valkey or Redis OSS cluster node type using the console:**

1. Sign in to the Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. From the list of clusters, choose the cluster you want to migrate.
4. Choose **Actions** and then choose **Modify**.
5. Choose the new node type from the node type list.
6. If you want to perform the migration process right away, choose **Apply immediately**. If **Apply immediately** is not chosen, the migration process is performed during the cluster's next maintenance window.
7. Choose **Modify**. If you chose **Apply immediately** in the previous step, the cluster's status changes to **modifying**. When the status changes to **available**, the modification is complete and you can begin using the new cluster.

*To modify a Valkey or Redis OSS cluster node type using the AWS CLI:*

Use the modify-replication-group API as shown following:

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group /
 --replication-group-id my-replication-group /
 --cache-node-type new-node-type /
 --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
 --replication-group-id my-replication-group ^
 --cache-node-type new-node-type ^
 --apply-immediately
```

In this scenario, the value of *new-node-type* is the node type you are migrating to. By passing the `--apply-immediately` parameter, the update will be applied immediately when the replication group transitions from **modifying** to **available** status. If **Apply immediately** is not chosen, the migration process is performed during the cluster's next maintenance window.

> **Note**
>
> If you are unable to modify the cluster with an `InvalidCacheClusterState` error, you need to remove a restore-failed node first.

#### Fixing or removing restore-failed-node(s)

The following procedure describes how to fix or remove restore-failed node(s) from your Valkey or Redis OSS cluster. To learn more on how ElastiCache node(s) enter a restore-failed state, see Viewing ElastiCache Node Status. We recommend first removing any nodes in a restore-failed state, then migrating the remaining previous generation nodes in the ElastiCache cluster to a newer generation node type, and finally adding back the required number of nodes.

To remove restore-failed node (console):

1. Sign in to the Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.

3. From the list of clusters, choose the cluster you want to remove a node from.
4. From the list of shards, choose the shard you want to remove a node from. Skip this step if cluster mode is disabled for the cluster.
5. From the list of nodes, choose the node with a status of `restore-failed`.
6. Choose **Actions** and then choose **Delete node**.

Once you remove the restore-failed node(s) from your ElastiCache cluster, you can now migrate to a newer generation type. For more information, see above on Migrating nodes on a Valkey or Redis OSS cluster.

To add back nodes to your ElastiCache cluster, see Adding nodes to an ElastiCache cluster.

### Migrating nodes on a Memcached cluster

To migrate ElastiCache for Memcached to a different node type, you must create a new cluster, which always starts out empty that your application can populate.

**To migrate your ElastiCache for Memcached cluster node type using the ElastiCache Console:**

- Create a new cluster with the new node type. For more information, see Creating a Memcached cluster (console).
- In your application, update the endpoints to the new cluster's endpoints. For more information, see Finding a Cluster's Endpoints (Console) (Memcached)
- Delete the old cluster. For more information, see Deleting a cluster in ElastiCache

## Managing clusters in ElastiCache

A *cluster* is a collection of one or more cache nodes, all of which run an instance of the Valkey, Memcached, and Redis OSS engine software. When you create a cluster, you specify the engine and version for all of the nodes to use.

#### Valkey and Redis OSS clusters

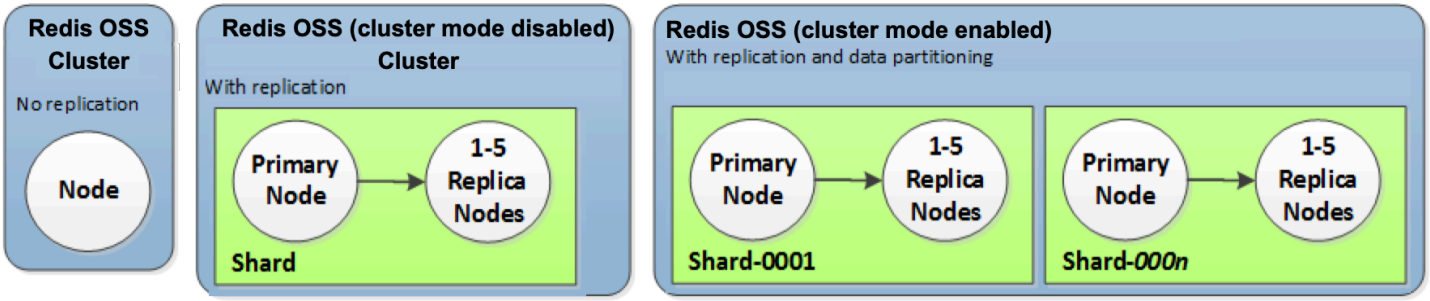
The following diagram illustrates a typical Valkey or Redis OSS cluster. These clusters can contain a single node or up to six nodes inside a shard (API/CLI: node group), A single-node Valkey or Redis OSS (cluster mode disabled) cluster has no shard, and a multi-node Valkey or Redis OSS (cluster mode disabled) cluster has a single shard. Valkey or Redis OSS (cluster mode enabled) clusters

can have up to 500 shards, with your data partitioned across the shards. The node or shard limit can be increased to a maximum of 500 per cluster if the engine version is Valkey 7.2 and higher or Redis OSS 5.0.6 and higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group. For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

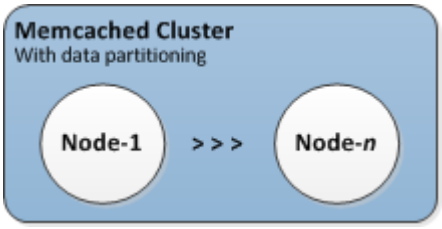
When you have multiple nodes in a Valkey or Redis OSS shard, one of the nodes is a read/write primary node. All other nodes in the shard are read-only replicas.

Typical Valkey or Redis OSS clusters look as follows.




### Memcached clusters

Typical Memcached clusters look as follows. Memcached clusters contain from 1 to 60 nodes, across which you horizontally partition your data.




### Elasticache operations for Valkey, Memcached, and Redis OSS

Most ElastiCache operations are performed at the cluster level. You can set up a cluster with a specific number of nodes and a parameter group that controls the properties for each node. All nodes within a cluster are designed to be of the same node type and have the same parameter and security group settings.

Every cluster must have a cluster identifier. The cluster identifier is a customer-supplied name for the cluster. This identifier specifies a particular cluster when interacting with the ElastiCache API and AWS CLI commands. The cluster identifier must be unique for that customer in an AWS Region.

ElastiCache supports multiple engine versions. Unless you have specific reasons, we recommend using the latest version.

ElastiCache clusters are designed to be accessed using an Amazon EC2 instance. If you launch your cluster in a virtual private cloud (VPC) based on the Amazon VPC service, you can access it from outside AWS. For more information, see Accessing ElastiCache resources from outside AWS.

For a list of supported versions, see Engine versions and upgrading in ElastiCache, Supported Redis OSS engine versions, and Supported ElastiCache for Memcached versions.

# Choosing a network type in ElastiCache

ElastiCache supports the Internet Protocol versions 4 and 6 (IPv4 and IPv6), allowing you to configure your cluster to accept:

- only IPv4 connections,
- only IPv6 connections,
- both IPv4 and IPv6 connections (dual-stack)

IPv6 is supported for workloads using Valkey 7.2 and onward, or Redis OSS 6.2 and onward, on all instances built on the Nitro system. There are no additional charges for accessing ElastiCache over IPv6.

> **Note**
>
> Migration of clusters created prior to the availability of IPV6 / dual-stack is not supported. Switching between network types on newly created clusters is also not supported.

IPv6 is supported for workloads using Memcached 1.6.6 onward on all instances built on the Nitro system. There are no additional charges for accessing ElastiCache over IPv6.

## Configuring subnets for network type

If you create a cluster in an Amazon VPC, you must specify a subnet group. ElastiCache uses that subnet group to choose a subnet and IP addresses within that subnet to associate with your nodes. ElastiCache clusters require a dual-stack subnet with both IPv4 and IPv6 addresses assigned to them to operate in dual-stack mode and an IPv6-only subnet to operate as IPv6-only.

## Using dual-stack

When using ElastiCache for Redis OSS in cluster mode enabled, from an application's perspective, connecting to all the cluster nodes through the configuration endpoint is no different than connecting directly to an individual cache node. To achieve this, a cluster-aware client must engage in a cluster discovery process and request the configuration information for all nodes. Redis' discovery protocol supports only one IP per node.

When you create a cluster with ElastiCache for Memcachedand choose dual-stack as the network type, you then need to designate an IP discovery type – either IPv4 or IPv6. ElastiCache will default

the network type and IP discovery to IPv6, but that can be changed. If you use Auto Discovery, only the IP addresses of your chosen IP type are returned returned to the Memcached client. For more information, see Automatically identify nodes in your cluster (Memcached).

To maintain backwards compatibility with all existing clients, IP discovery is introduced, which allows you to select the IP type (i.e., IPv4 or IPv6) to advertise in the discovery protocol. While this limits auto discovery to only one IP type, dual-stack is still beneficial for cluster mode enabled workloads, as it enables migrations (or rollbacks) from an IPv4 to an IPv6 Discovery IP type with no downtime.

### TLS enabled dual stack ElastiCache clusters

When TLS is enabled for ElastiCache clusters the cluster discovery functions such as `cluster slots`, `cluster shards`, and `cluster nodes` with Valkey or Redis OSS and `config get cluster` with Memcached return hostnames instead of IPs. The hostnames are then used instead of IPs to connect to the ElastiCache cluster and perform a TLS handshake. This means that clients won't be affected by the IP Discovery parameter. *For TLS enabled clusters the IP Discovery parameter has no effect on the preferred IP protocol.* Instead, the IP protocol used will be determined by which IP protocol the client prefers when resolving DNS hostnames.

For examples on how to configure an IP protocol preference when resolving DNS hostnames, see TLS enabled dual stack ElastiCache clusters.

### Using the AWS Management Console (Valkey and Redis OSS)

When creating a cluster using the AWS Management Console, under **Connectivity**, choose a network type, either **IPv4**, **IPv6** or **Dual stack**. If you are creating a Valkey or Redis OSS (cluster mode enabled) cluster and choose dual stack, you then must select a **Discovery IP type**, either IPv6 or IPv4.

For more information, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console) or Creating a Valkey or Redis OSS (cluster mode disabled) (Console).

When creating a replication group using the AWS Management Console, choose a network type, either **IPv4**, **IPv6** or **Dual stack**. If you choose dual stack, you then must select a **Discovery IP type**, either IPv6 or IPv4.

For more information, see Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch or Creating a replication group in Valkey or Redis OSS (Cluster Mode Enabled) from scratch.

## Using the AWS Management Console (Memcached)

When creating a cluster using the AWS Management Console, under **Connectivity**, choose a network type, either **IPv4**, **IPv6** or **Dual stack**. If you choose dual stack, you then must select a **Discovery IP type**, either IPv6 or IPv4.

For more information, see Creating a Memcached cluster (console).

## Using the CLI with Valkey, Memcached, or Redis OSS.

### Redis OSS

When creating a cluster with Valkey or Redis OSS using the CLI, you use the create-cache-cluster command and specify the `NetworkType` and `IPDiscovery` parameters:

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
    --cache-cluster-id "cluster-test" \
    --engine redis \
    --cache-node-type cache.m5.large \
    --num-cache-nodes 1 \
    --network-type dual_stack \
    --ip-discovery ipv4
```

For Windows:

```
aws elasticache create-cache-cluster ^
    --cache-cluster-id "cluster-test" ^
    --engine redis ^
    --cache-node-type cache.m5.large ^
    --num-cache-nodes 1 ^
    --network-type dual_stack ^
    --ip-discovery ipv4
```

When creating a replication group with cluster mode disabled using the CLI, you use the create-replication-group command and specify the `NetworkType` and `IPDiscovery` parameters:

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
```

```
   --replication-group-id sample-repl-group \
   --replication-group-description "demo cluster with replicas" \
   --num-cache-clusters 3 \
   --primary-cluster-id redis01 \
   --network-type dual_stack \
   --ip-discovery ipv4
```

For Windows:

```
aws elasticache create-replication-group ^
   --replication-group-id sample-repl-group ^
   --replication-group-description "demo cluster with replicas" ^
   --num-cache-clusters 3 ^
   --primary-cluster-id redis01 ^
   --network-type dual_stack ^
   --ip-discovery ipv4
```

When creating a replication group with cluster mode enabled and use IPv4 for IP discovery using the CLI, you use the create-replication-group command and specify the `NetworkType` and `IPDiscovery` parameters:

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
  --replication-group-id demo-cluster \
  --replication-group-description "demo cluster" \
  --cache-node-type cache.m5.large \
  --num-node-groups 2 \
  --engine redis \
  --cache-subnet-group-name xyz \
  --network-type dual_stack \
  --ip-discovery ipv4 \
  --region us-east-1
```

For Windows:

```
aws elasticache create-replication-group ^
  --replication-group-id demo-cluster ^
  --replication-group-description "demo cluster" ^
  --cache-node-type cache.m5.large ^
  --num-node-groups 2 ^
```

```
  --engine redis ^
  --cache-subnet-group-name xyz ^
  --network-type dual_stack ^
  --ip-discovery ipv4 ^
  --region us-east-1
```

When creating a replication group with cluster mode enabled and use IPv6 for IP discovery using the CLI, you use the create-replication-group command and specify the `NetworkType` and `IPDiscovery` parameters:

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
  --replication-group-id demo-cluster \
  --replication-group-description "demo cluster" \
  --cache-node-type cache.m5.large \
  --num-node-groups 2 \
  --engine redis \
  --cache-subnet-group-name xyz \
  --network-type dual_stack \
  --ip-discovery ipv6 \
  --region us-east-1
```

For Windows:

```
aws elasticache create-replication-group ^
  --replication-group-id demo-cluster ^
  --replication-group-description "demo cluster" ^
  --cache-node-type cache.m5.large ^
  --num-node-groups 2 ^
  --engine redis ^
  --cache-subnet-group-name xyz ^
  --network-type dual_stack ^
  --ip-discovery ipv6 ^
  --region us-east-1
```

**Memcached**

When creating a cluster with Memcached using the CLI, you use the create-cache-cluster command and specify the `NetworkType` and `IPDiscovery` parameters:

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
    --cache-cluster-id "cluster-test" \
    --engine memcached \
    --cache-node-type cache.m5.large \
    --num-cache-nodes 1 \
    --network-type dual_stack \
    --ip-discovery ipv4
```

For Windows:

```
aws elasticache create-cache-cluster ^
    --cache-cluster-id "cluster-test" ^
    --engine memcached ^
    --cache-node-type cache.m5.large ^
    --num-cache-nodes 1 ^
    --network-type dual_stack ^
    --ip-discovery ipv4
```

## Automatically identify nodes in your cluster (Memcached)

For clusters running the Memcached engine, ElastiCache supports *Auto Discovery*—the ability for client programs to automatically identify all of the nodes in a cluster, and to initiate and maintain connections to all of these nodes.

> **Note**
>
> Auto Discovery is added for clusters running on Amazon ElastiCache Memcached. Auto Discovery is not available for Valkey or Redis OSS engines.

With Auto Discovery, your application does not need to manually connect to individual cache nodes; instead, your application connects to one Memcached node and retrieves the list of nodes. From that list your application is aware of the rest of the nodes in the cluster and can connect to any of them. You do not need to hard code the individual cache node endpoints in your application.

If you are using dual stack network type on your cluster, Auto Discovery will return only IPv4 or IPv6 addresses, depending on which one you select. For more information, see Choosing a network type in ElastiCache .

All of the cache nodes in the cluster maintain a list of metadata about all of the other nodes. This metadata is updated whenever nodes are added or removed from the cluster.

**Topics**


- Benefits of Auto Discovery with Memcached
- How Auto Discovery Works
- Using Auto Discovery
- Connecting to Memcached Cache Nodes Manually
- Adding Auto Discovery to your Memcached client library
- ElastiCache clients with auto discovery

## Benefits of Auto Discovery with Memcached

When using Memcached, Auto Discovery offers the following benefits:

- When you increase the number of nodes in a cluster, the new nodes register themselves with the configuration endpoint and with all of the other nodes. When you remove nodes from the cache cluster, the departing nodes deregister themselves. In both cases, all of the other nodes in the cluster are updated with the latest cache node metadata.
- Cache node failures are automatically detected; failed nodes are automatically replaced.

  > **Note**
  >
  > Until node replacement completes, the node will continue to fail.

- A client program only needs to connect to the configuration endpoint. After that, the Auto Discovery library connects to all of the other nodes in the cluster.
- Client programs poll the cluster once per minute (this interval can be adjusted if necessary). If there are any changes to the cluster configuration, such as new or deleted nodes, the client receives an updated list of metadata. Then the client connects to, or disconnects from, these nodes as needed.

Auto Discovery is enabled on all ElastiCache Memcached clusters. You do not need to reboot any of your cache nodes to use this feature.

## How Auto Discovery Works

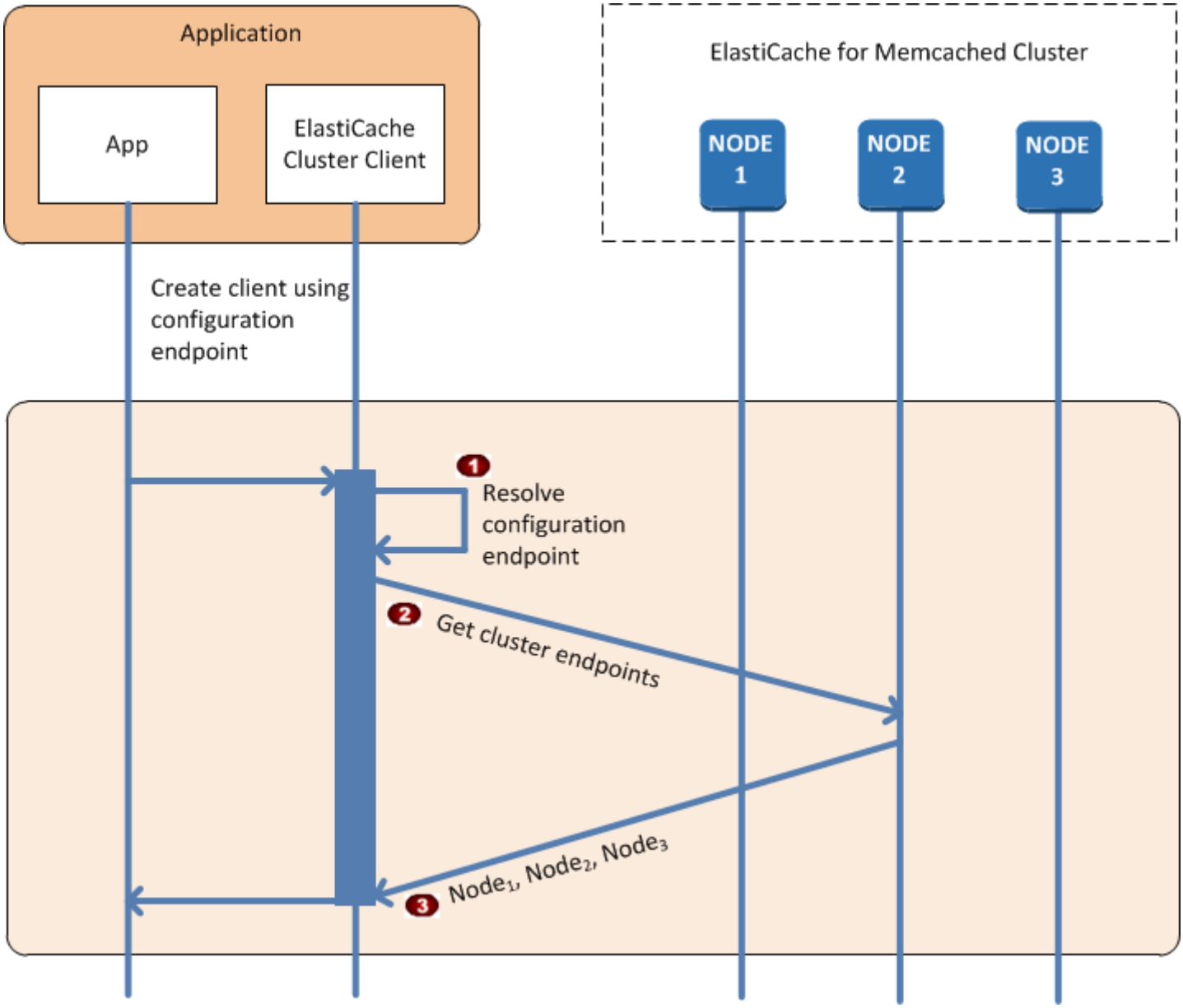
### Topics


- Connecting to Cache Nodes
- Normal Cluster Operations
- Other Operations


This section describes how client applications use the ElastiCache Cluster Client to manage cache node connections, and interact with data items in the cache.

### Connecting to Cache Nodes

From the application's point of view, connecting to the cluster configuration endpoint is no different from connecting directly to an individual cache node. The following sequence diagram shows the process of connecting to cache nodes.




**Process of Connecting to Cache Nodes**

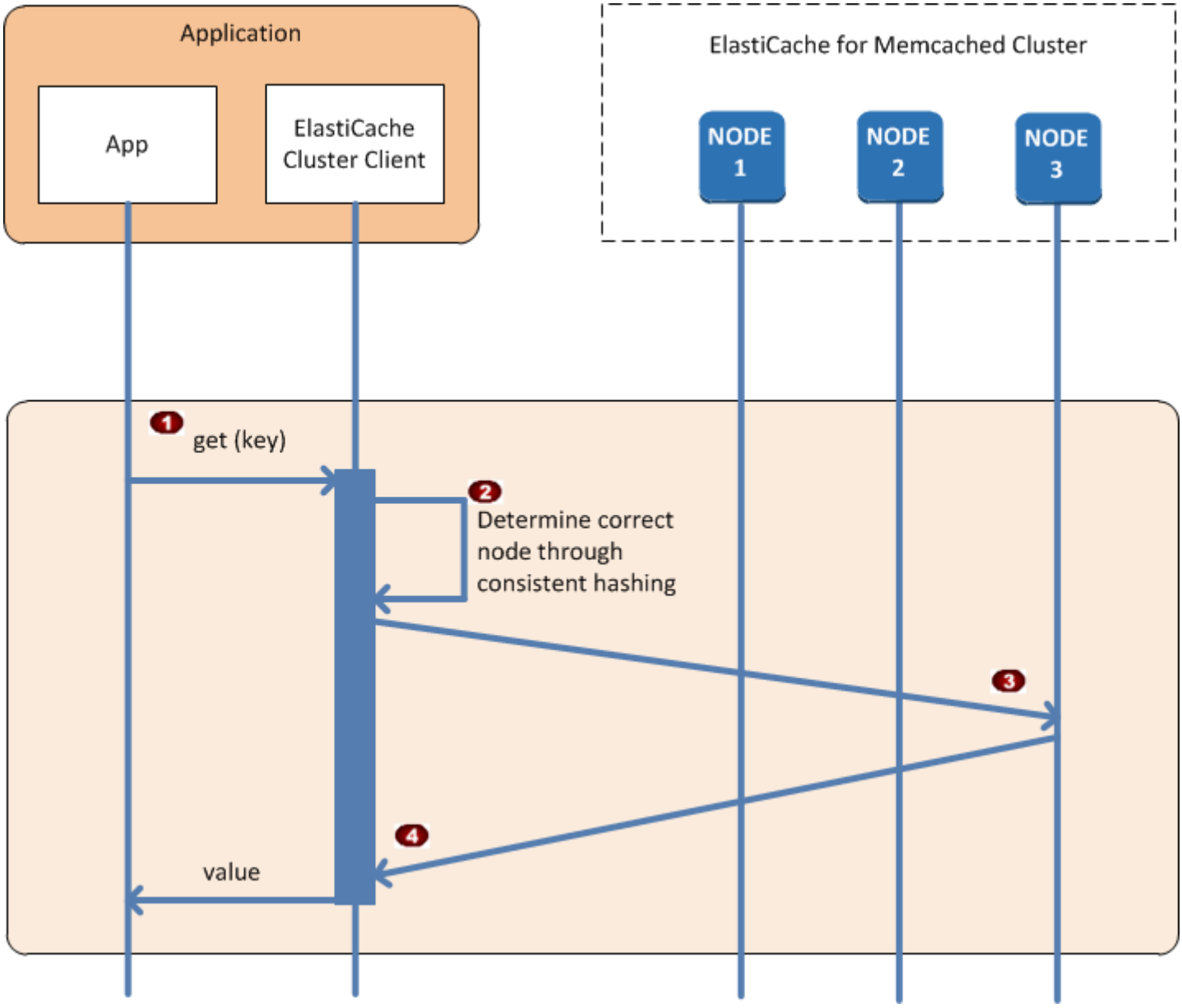
- The application resolves the configuration endpoint's DNS name. Because the configuration endpoint maintains CNAME entries for all of the cache nodes, the DNS name resolves to one of the nodes; the client can then connect to that node.
- The client requests the configuration information for all of the other nodes. Since each node maintains configuration information for all of the nodes in the cluster, any node can pass configuration information to the client upon request.

- The client receives the current list of cache node hostnames and IP addresses. It can then connect to all of the other nodes in the cluster.

> **Note**
>
> The client program refreshes its list of cache node hostnames and IP addresses once per minute. This polling interval can be adjusted if necessary.

### Normal Cluster Operations

When the application has connected to all of the cache nodes, ElastiCache Cluster Client determines which nodes should store individual data items, and which nodes should be queried for those data items later. The following sequence diagram shows the process of normal cluster operations.




**Process of Normal Cluster Operations**

- The application issues a *get* request for a particular data item, identified by its key.
- The client uses a hashing algorithm against the key to determine which cache node contains the data item.
- The data item is requested from the appropriate node.
- The data item is returned to the application.

### Other Operations

In some situations, you might make a change to a cluster's nodes. For example, you might add an additional node to accommodate additional demand, or delete a node to save money during periods of reduced demand. Or you might replace a node due to a node failure of one sort or another.

When there is a change in the cluster that requires a metadata update to the cluster's endpoints, that change is made to all nodes at the same time. Thus the metadata in any given node is consistent with the metadata in all of the other nodes in the cluster.

In each of these cases, the metadata is consistent among all the nodes at all times since the metadata is updated at the same time for all nodes in the cluster. You should always use the configuration endpoint to obtain the endpoints of the various nodes in the cluster. By using the configuration endpoint, you ensure that you will not be obtaining endpoint data from a node that “disappears” on you.

#### Adding a Node

During the time that the node is being spun up, its endpoint is not included in the metadata. As soon as the node is available, it is added to the metadata of each of the cluster’s nodes. In this scenario, the metadata is consistent among all the nodes and you will be able to interact with the new node only after it is available. Before the node being available, you will not know about it and will interact with the nodes in your cluster the same as though the new node does not exist.

#### Deleting a Node

When a node is removed, its endpoint is first removed from the metadata and then the node is removed from the cluster. In this scenario the metadata in all the nodes is consistent and there is no time in which it will contain the endpoint for the node to be removed while the node is not available. During the node removal time it is not reported in the metadata and so your application will only be interacting with the n-1 remaining nodes, as though the node does not exist.

#### Replacing a Node

If a node fails, ElastiCache takes down that node and spins up a replacement. The replacement process takes a few minutes. During this time the metadata in all the nodes still shows the endpoint for the failed node, but any attempt to interact with the node will fail. Therefore, your logic should always include retry logic.

## Using Auto Discovery

To begin using Auto Discovery with ElastiCache for Memcached, follow these steps:

- Obtain the Configuration Endpoint
- Download the ElastiCache Cluster Client
- Modify Your Application Program

### Obtain the Configuration Endpoint

To connect to a cluster, client programs must know the cluster configuration endpoint. See the topic Finding a Cluster's Endpoints (Console) (Memcached)

You can also use the `aws elasticache describe-cache-clusters` command with the `--show-cache-node-info` parameter:

Whatever method you use to find the cluster's endpoints, the configuration endpoint will always have **.cfg** in its address.

#### Example Finding endpoints using the AWS CLI for ElastiCache

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-clusters \
    --cache-cluster-id mycluster \
    --show-cache-node-info
```

For Windows:

```
aws elasticache describe-cache-clusters ^
    --cache-cluster-id mycluster ^
    --show-cache-node-info
```

This operation produces output similar to the following (JSON format):

```
{
    "CacheClusters": [
        {
            "Engine": "memcached",
            "CacheNodes": [
                {
```

```
                    "CacheNodeId": "0001",
                    "Endpoint": {
                        "Port": 11211,
                        "Address": "mycluster.fnjyzo.cfg.0001.use1.cache.amazonaws.com"
                    },
                    "CacheNodeStatus": "available",
                    "ParameterGroupStatus": "in-sync",
                    "CacheNodeCreateTime": "2016-10-12T21:39:28.001Z",
                    "CustomerAvailabilityZone": "us-east-1e"
                },
                {
                    "CacheNodeId": "0002",
                    "Endpoint": {
                        "Port": 11211,
                        "Address": "mycluster.fnjyzo.cfg.0002.use1.cache.amazonaws.com"
                    },
                    "CacheNodeStatus": "available",
                    "ParameterGroupStatus": "in-sync",
                    "CacheNodeCreateTime": "2016-10-12T21:39:28.001Z",
                    "CustomerAvailabilityZone": "us-east-1a"
                }
            ],
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "CacheParameterGroupName": "default.memcached1.4",
                "ParameterApplyStatus": "in-sync"
            },
            "CacheClusterId": "mycluster",
            "PreferredAvailabilityZone": "Multiple",
            "ConfigurationEndpoint": {
                "Port": 11211,
                "Address": "mycluster.fnjyzo.cfg.use1.cache.amazonaws.com"
            },
            "CacheSecurityGroups": [],
            "CacheClusterCreateTime": "2016-10-12T21:39:28.001Z",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterStatus": "available",
            "NumCacheNodes": 2,
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "CacheSubnetGroupName": "default",
            "EngineVersion": "1.4.24",
            "PendingModifiedValues": {},
            "PreferredMaintenanceWindow": "sat:06:00-sat:07:00",
```

```
            "CacheNodeType": "cache.r5.large"
        }
    ]
}
```

### Download the ElastiCache Cluster Client

To take advantage of Auto Discovery, client programs must use the *ElastiCache Cluster Client*. The ElastiCache Cluster Client is available for Java, PHP, and .NET and contains all of the necessary logic for discovering and connecting to all of your cache nodes.

### To download the ElastiCache Cluster Client

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the ElastiCache console, choose **ElastiCache Cluster Client** then choose **Download**.

The source code for the ElastiCache Cluster Client for Java is available at https://github.com/amazonwebservices/aws-elasticache-cluster-client-memcached-for-java. This library is based on the popular Spymemcached client. The ElastiCache Cluster Client is released under the Amazon Software License https://aws.amazon.com/asl. You are free to modify the source code as you see fit. You can even incorporate the code into other open source Memcached libraries, or into your own client code.

> **Note**
>
> To use the ElastiCache Cluster Client for PHP, you will first need to install it on your Amazon EC2 instance. For more information, see Installing the ElastiCache cluster client for PHP.
> For a TLS supported client download the binary with PHP version 7.4 or higher.
> To use the ElastiCache Cluster Client for .NET, you will first need to install it on your Amazon EC2 instance. For more information, see Installing the ElastiCache cluster client for .NET.

### Modify Your Application Program

Modify your application program so that it uses Auto Discovery. The following sections show how to use the ElastiCache Cluster Client for Java, PHP, and .NET.

**Important**

When specifying the cluster's configuration endpoint, be sure that the endpoint has ".cfg" in its address as shown here. Do not use a CNAME or an endpoint without ".cfg" in it.

```
"mycluster.fnjyzo.cfg.use1.cache.amazonaws.com";
```

Failure to explicitly specify the cluster's configuration endpoint results in configuring to a specific node.

### Using the ElastiCache Cluster Client for Java

The program below demonstrates how to use the ElastiCache Cluster Client to connect to a cluster configuration endpoint and add a data item to the cache. Using Auto Discovery, the program connects to all of the nodes in the cluster without any further intervention.

```
package com.amazon.elasticache;

import java.io.IOException;
import java.net.InetSocketAddress;

// Import the &AWS;-provided library with Auto Discovery support
import net.spy.memcached.MemcachedClient;

public class AutoDiscoveryDemo {

    public static void main(String[] args) throws IOException {

        String configEndpoint = "mycluster.fnjyzo.cfg.use1.cache.amazonaws.com";
        Integer clusterPort = 11211;

        MemcachedClient client = new MemcachedClient(
                                 new InetSocketAddress(configEndpoint,
                                                       clusterPort));
        // The client will connect to the other cache nodes automatically.

        // Store a data item for an hour.
        // The client will decide which cache host will store this item.
        client.set("theKey", 3600, "This is the data value");
    }
```

```
}
```

ElastiCache Serverless caches always have in-transit encryption (TLS) enabled. You can also enable TLS on node-based clusters. To connect to a TLS-enabled cache, configure the client with an SSLContext as shown in the following example.

> **Note**
>
> The `setSSLContext` method is only available in the AWS fork of the spymemcached client. For the source, see aws-elasticache-cluster-client-memcached-for-java on the GitHub website. This method is not part of the standard open-source spymemcached library.

```
import java.security.KeyStore;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;
import net.spy.memcached.AddrUtil;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.MemcachedClient;
public class TLSDemo {
    public static void main(String[] args) throws Exception {
        ConnectionFactoryBuilder connectionFactoryBuilder = new
 ConnectionFactoryBuilder();
        // Build SSLContext
        TrustManagerFactory tmf =
 TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init((KeyStore) null);
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, tmf.getTrustManagers(), null);
        // Create the client in TLS mode
        connectionFactoryBuilder.setSSLContext(sslContext);
        MemcachedClient client = new MemcachedClient(connectionFactoryBuilder.build(),
 AddrUtil.getAddresses("mycluster.fnjyzo.cfg.use1.cache.amazonaws.com:11211"));

        // Store a data item for an hour.
        client.set("theKey", 3600, "This is the data value");
    }
}
```

For more information about connecting with TLS, see Creating a TLS Memcached client using Java. For a Serverless cache, use the cache's Serverless endpoint on port 11211. The Serverless endpoint does not contain ".cfg" in the address. For a node-based cluster with TLS enabled, use the configuration endpoint that contains ".cfg".

### Using the ElastiCache Cluster Client for PHP

The program below demonstrates how to use the ElastiCache Cluster Client to connect to a cluster configuration endpoint and add a data item to the cache. Using Auto Discovery, the program will connect to all of the nodes in the cluster without any further intervention.

To use the ElastiCache Cluster Client for PHP, you will first need to install it on your Amazon EC2 instance. For more information, see Installing the ElastiCache cluster client for PHP

```
<?php

 /**
  * Sample PHP code to show how to integrate with the Amazon ElastiCache
  * Auto Discovery feature.
  */

  /* Configuration endpoint to use to initialize memcached client.
   * This is only an example.  */
  $server_endpoint = "mycluster.fnjyzo.cfg.use1.cache.amazonaws.com";

  /* Port for connecting to the ElastiCache cluster.
   * This is only an example  */
  $server_port = 11211;

 /**
  * The following will initialize a Memcached client to utilize the Auto Discovery
 feature.
  *
  * By configuring the client with the Dynamic client mode with single endpoint, the
  * client will periodically use the configuration endpoint to retrieve the current
 cache
  * cluster configuration. This allows scaling the cluster up or down in number of
 nodes
  * without requiring any changes to the PHP application.
  *
  * By default the Memcached instances are destroyed at the end of the request.
  * To create an instance that persists between requests,
  *    use persistent_id to specify a unique ID for the instance.
```

```
 * All instances created with the same persistent_id will share the same connection.
 * See http://php.net/manual/en/memcached.construct.php for more information.
 */
 $dynamic_client = new Memcached('persistent-id');
 $dynamic_client->setOption(Memcached::OPT_CLIENT_MODE,
Memcached::DYNAMIC_CLIENT_MODE);
 $dynamic_client->addServer($server_endpoint, $server_port);

 /**
 * Store the data for 60 seconds in the cluster.
 * The client will decide which cache host will store this item.
 */
 $dynamic_client->set('key', 'value', 60);


/**
 * Configuring the client with Static client mode disables the usage of Auto Discovery
 * and the client operates as it did before the introduction of Auto Discovery.
 * The user can then add a list of server endpoints.
 */
 $static_client = new Memcached('persistent-id');
 $static_client->setOption(Memcached::OPT_CLIENT_MODE, Memcached::STATIC_CLIENT_MODE);
 $static_client->addServer($server_endpoint, $server_port);

/**
 * Store the data without expiration.
 * The client will decide which cache host will store this item.
 */
 $static_client->set('key', 'value');
 ?>
```

For an example on how to use the ElastiCache Cluster Client with TLS enabled, see Using in transit encryption with PHP and Memcached.

### Using the ElastiCache Cluster Client for .NET

> **Note**
>
> The ElastiCache .NET cluster client has been deprecated as of May, 2022.

.NET client for ElastiCache is open source at https://github.com/awslabs/elasticache-cluster-config-net.

.NET applications typically get their configurations from their config file. The following is a sample application config file.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSections>
        <section
            name="clusterclient"
            type="Amazon.ElastiCacheCluster.ClusterConfigSettings,
 Amazon.ElastiCacheCluster" />
    </configSections>

    <clusterclient>
        <!-- the hostname and port values are from step 1 above -->
        <endpoint hostname="mycluster.fnjyzo.cfg.use1.cache.amazonaws.com"
 port="11211" />
    </clusterclient>
</configuration>
```

The C# program below demonstrates how to use the ElastiCache Cluster Client to connect to a cluster configuration endpoint and add a data item to the cache. Using Auto Discovery, the program will connect to all of the nodes in the cluster without any further intervention.

```
// *****************
// Sample C# code to show how to integrate with the Amazon ElastiCcache Auto Discovery
 feature.

using System;

using Amazon.ElastiCacheCluster;

using Enyim.Caching;
using Enyim.Caching.Memcached;

public class DotNetAutoDiscoveryDemo  {

    public static void Main(String[] args)  {

        // instantiate a new client.
        ElastiCacheClusterConfig config = new ElastiCacheClusterConfig();
        MemcachedClient memClient = new MemcachedClient(config);

        // Store the data for 3600 seconds (1hour) in the cluster.
```

```
        // The client will decide which cache host will store this item.
        memClient.Store(StoreMode.Set, 3600, "This is the data value.");

    }  // end Main

}  // end class DotNetAutoDiscoverDemo
```

## Connecting to Memcached Cache Nodes Manually

If your client program does not use Auto Discovery, it can manually connect to each of the Memcached cache nodes. This is the default behavior for Memcached clients.

You can obtain a list of cache node hostnames and port numbers from the AWS Management Console. You can also use the AWS CLI `aws elasticache describe-cache-clusters` command with the `--show-cache-node-info` parameter.

### Example

The following Java code snippet shows how to connect to all of the nodes in a four-node cluster:

```
...

ArrayList<String> cacheNodes = new ArrayList<String>(
 Arrays.asList(
     "mycachecluster.fnjyzo.0001.use1.cache.amazonaws.com:11211",
     "mycachecluster.fnjyzo.0002.use1.cache.amazonaws.com:11211",
     "mycachecluster.fnjyzo.0003.use1.cache.amazonaws.com:11211",
     "mycachecluster.fnjyzo.0004.use1.cache.amazonaws.com:11211"));

MemcachedClient cache = new MemcachedClient(AddrUtil.getAddresses(cacheNodes));

...
```

**Important**

If you scale up or scale down your cluster by adding or removing nodes, you will need to update the list of nodes in the client code.

## Adding Auto Discovery to your Memcached client library

The configuration information for Auto Discovery is stored redundantly in each Memcached cluster node. Client applications can query any cache node and obtain the configuration information for all of the nodes in the cluster.

The way in which an application does this depends upon the cache engine version:

- If the cache engine version is **1.4.14 or higher**, use the `config` command.
- If the cache engine version is **lower than 1.4.14**, use the `get AmazonElastiCache:cluster` command.

The outputs from these two commands are identical, and are described in the Output Format section below.

### Cache engine version 1.4.14 or higher

For cache engine version 1.4.14 or higher, use the `config` command. This command has been added to the Memcached ASCII and binary protocols by ElastiCache, and is implemented in the ElastiCache Cluster Client. If you want to use Auto Discovery with another client library, then that library will need to be extended to support the `config` command.

> **Note**
>
> The following documentation pertains to the ASCII protocol; however, the `config` command supports both ASCII and binary. If you want to add Auto Discovery support using the binary protocol, refer to the source code for the ElastiCache Cluster Client.

#### Syntax

```
config [sub-command] [key]
```

#### Options

| Name | Description | Required |
| --- | --- | --- |
| `sub-command` | The sub-command used to interact with a cache node. For Auto Discovery, this sub-command is `get`. | Yes |

| Name | Description | Required |
| --- | --- | --- |
| `key` | The key under which the cluster configuration is stored. For Auto Discovery, this key is named `cluster`. | Yes |

To get the cluster configuration information, use the following command:

```
config get cluster
```

**Cache engine version 1.4.14 or lower**

To get the cluster configuration information, use the following command:

```
get AmazonElastiCache:cluster
```

> **Note**
>
> Do not tamper with the "AmazonElastiCache:cluster" key, since this is where the cluster configuration information resides. If you do overwrite this key, then the client may be incorrectly configured for a brief period of time (no more than 15 seconds) before ElastiCache automatically and correctly updates the configuration information.

**Output Format**

Whether you use `config get cluster` or `get AmazonElastiCache:cluster`, the reply consists of two lines:

- The version number of the configuration information. Each time a node is added or removed from the cluster, the version number increases by one.
- A list of cache nodes. Each node in the list is represented by a *hostname|ip-address|port* group, and each node is delimited by a space.

A carriage return and a linefeed character (CR + LF) appears at the end of each line. The data line contains a linefeed character (LF) at the end, to which the CR + LF is added. The config version line is terminated by LF without the CR.

A cluster containing three nodes would be represented as follows:

```
configversion\n
hostname|ip-address|port hostname|ip-address|port hostname|ip-address|port\n\r\n
```

Each node is shown with both the CNAME and the private IP address. The CNAME will always be present; if the private IP address is not available, it will not be shown; however, the pipe characters "|" will still be printed.

**Example**

Here is an example of the payload returned when you query the configuration information:

```
CONFIG cluster 0 136\r\n
12\n
myCluster.pc4ldq.0001.use1.cache.amazonaws.com|10.82.235.120|11211
 myCluster.pc4ldq.0002.use1.cache.amazonaws.com|10.80.249.27|11211\n\r\n
END\r\n
```

**Note**

- The second line indicates that the configuration information has been modified twelve times so far.
- In the third line, the list of nodes is in alphabetical order by hostname. This ordering might be in a different sequence from what you are currently using in your client application.

## ElastiCache clients with auto discovery

Cluster client programs can automatically identify and connect to all cluster nodes that are running the Memcached engine.

This section discusses installing and configuring the ElastiCache PHP and .NET clients for use with auto discovery.

**Topics**

- Installing & compiling cluster clients
- Configuring ElastiCache clients

### Installing & compiling cluster clients

This section covers installing, configuring, and compiling the PHP and .NET Amazon ElastiCache auto discovery cluster clients.

#### Topics

- Installing the ElastiCache cluster client for .NET
- Installing the ElastiCache cluster client for PHP
- Compiling the source code for the ElastiCache cluster client for PHP

### Installing the ElastiCache cluster client for .NET

You can find the ElastiCache .NET Cluster Client code as open source at https://github.com/awslabs/elasticache-cluster-config-net.

This section describes how to install, update, and remove the .NET components for the ElastiCache Cluster Client on Amazon EC2 instances. For more information about auto discovery, see Automatically identify nodes in your cluster (Memcached). For sample .NET code to use the client, see Using the ElastiCache Cluster Client for .NET.

#### Topics

- Installing .NET
- Download the ElastiCache .NET cluster client for ElastiCache
- Install AWS assemblies with NuGet

### Installing .NET

You must have .NET 3.5 or later installed to use the AWS .NET SDK for ElastiCache. If you don't have .NET 3.5 or later, you can download and install the latest version from http://www.microsoft.com/net.

### Download the ElastiCache .NET cluster client for ElastiCache

#### To download the ElastiCache .NET cluster client

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. On the navigation pane, click **ElastiCache Cluster Client**.
3. In the **Download ElastiCache Memcached Cluster Client** list, select **.NET**, and then click **Download.**

### Install AWS assemblies with NuGet

NuGet is a package management system for the .NET platform. NuGet is aware of assembly dependencies and installs all required files automatically. NuGet installed assemblies are stored with your solution, rather than in a central location such as `Program Files`, so you can install versions specific to an application without creating compatibility issues.

### Installing NuGet

NuGet can be installed from the Installation Gallery on MSDN; see https://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c. If you are using Visual Studio 2010 or later, NuGet is automatically installed.

You can use NuGet from either **Solution Explorer** or **Package Manager Console**.

### Using NuGet from Solution Explorer

#### To use NuGet from Solution Explorer in Visual Studio 2010

1. From the **Tools** menu, select **Library Package Manager**.
2. Click **Package Manager Console**.

#### To use NuGet from Solution Explorer in Visual Studio 2012 or Visual Studio 2013

1. From the **Tools** menu, select **NuGet Package Manager**.
2. Click **Package Manager Console**.

From the command line, you can install the assemblies using `Install-Package`, as shown following.

```
Install-Package Amazon.ElastiCacheCluster
```

To see a page for every package that is available through NuGet, such as the AWSSDK and AWS.Extensions assemblies, see the NuGet website at http://www.nuget.org. The page for each

package includes a sample command line for installing the package using the console and a list of the previous versions of the package that are available through NuGet.

For more information on **Package Manager Console** commands, see http://nuget.codeplex.com/wikipage?title=Package%20Manager%20Console%20Command%20Reference%20%28v1.3%29.

### Installing the ElastiCache cluster client for PHP

This section describes how to install, update, and remove the PHP components for the ElastiCache Cluster Client on Amazon EC2 instances. For more information about Auto Discovery, see Automatically identify nodes in your cluster (Memcached). For sample PHP code to use the client. see Using the ElastiCache Cluster Client for PHP.

#### Topics


- Downloading the installation package
- For users who already have php-memcached extension installed
- Installation steps for new users
- Removing the PHP cluster client


### Downloading the installation package

To ensure that you use the correct version of the ElastiCache Cluster Client for PHP, you will need to know what version of PHP is installed on your Amazon EC2 instance. You will also need to know whether your Amazon EC2 instance is running a 64-bit or 32-bit version of Linux.

#### To determine the PHP version installed on your Amazon EC2 instance

- At the command prompt, run the following command:

  ```
  php -v
  ```

  The PHP version will be shown in the output, as in this example:

  ```
  PHP 5.4.10 (cli) (built: Jan 11 2013 14:48:57)
  Copyright (c) 1997-2012 The PHP Group
  Zend Engine v2.4.0, Copyright (c) 1998-2012 Zend Technologies
  ```

  **Note**

  If your PHP and Memcached versions are incompatible, you will get an error message something like the following:

  ```
  PHP Warning: PHP Startup: memcached: Unable to initialize module
  Module compiled with module API=20100525
  ```

```
PHP compiled with module API=20131226
These options need to match
in Unknown on line 0
```

If this happens, you need to compile the module from the source code. For more information, see Compiling the source code for the ElastiCache cluster client for PHP.

**To determine your Amazon EC2 AMI architecture (64-bit or 32-bit)**

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the **Instances** list, click your Amazon EC2 instance.
3. In the **Description** tab, look for the **AMI:** field. A 64-bit instance should have `x86_64` as part of the description; for a 32-bit instance, look for `i386` or `i686` in this field.

You are now ready to download the ElastiCache Cluster Client.

**To download the ElastiCache cluster client for PHP**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the ElastiCache console, choose **ElastiCache Cluster Client**.
3. From the **Download ElastiCache Memcached Cluster Client** list, choose the ElastiCache Cluster Client that matches your PHP version and AMI architecture, then choose the **Download** button.

**For users who already have *php-memcached* extension installed**

**To update the `php-memcached` installation**

1. Remove the previous installation of the Memcached extension for PHP as described by the topic Removing the PHP cluster client.
2. Install the new ElastiCache `php-memcached` extension as described previously in Installation steps for new users.

### Installation steps for new users

#### Topics

- Installing PHP 7.x for new users
- Installing PHP 5.x for new users

### Installing PHP 7.x for new users

#### Topics

- To install PHP 7 on a Ubuntu server 14.04 LTS AMI (64-bit and 32-bit)
- To install PHP 7 on an Amazon Linux 201609 AMI
- To install PHP 7 on an SUSE Linux AMI

### To install PHP 7 on a Ubuntu server 14.04 LTS AMI (64-bit and 32-bit)

1. Launch a new instance from the AMI.
2. Run the following commands:

```
sudo apt-get update
sudo apt-get install gcc g++
```

3. Install PHP 7.

```
sudo apt-get install php7.0
```

4. Download the Amazon ElastiCache Cluster Client.

```
wget https://elasticache-downloads.s3.amazonaws.com/ClusterClient/PHP-7.0/
latest-64bit
```

5. Extract `latest-64bit`.

```
tar -zxvf latest-64bit
```

6. With root permissions, copy the extracted artifact file `amazon-elasticache-cluster-client.so` into `/usr/lib/php/20151012`.

```
sudo mv artifact/amazon-elasticache-cluster-client.so /usr/lib/php/20151012
```

7. Insert the line `extension=amazon-elasticache-cluster-client.so` into the file `/etc/php/7.0/cli/php.ini`.

```
echo "extension=amazon-elasticache-cluster-client.so" | sudo tee --append /etc/
php/7.0/cli/php.ini
```

8. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

**To install PHP 7 on an Amazon Linux 201609 AMI**

1. Launch a new instance from the AMI.
2. Run the following command:

```
sudo yum install gcc-c++
```

3. Install PHP 7.

```
sudo yum install php70
```

4. Download the Amazon ElastiCache Cluster Client.

```
wget https://elasticache-downloads.s3.amazonaws.com/ClusterClient/PHP-7.0/
latest-64bit
```

5. Extract `latest-64bit`.

```
tar -zxvf latest-64bit
```

6. With root permission, copy the extracted artifact file `amazon-elasticache-cluster-client.so` into `/usr/lib64/php/7.0/modules/`.

```
sudo mv artifact/amazon-elasticache-cluster-client.so /usr/lib64/php/7.0/modules/
```

7. Create the `50-memcached.ini` file.

```
echo "extension=amazon-elasticache-cluster-client.so" | sudo tee --append /etc/
php-7.0.d/50-memcached.ini
```

8. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

**To install PHP 7 on an SUSE Linux AMI**

1. Launch a new instance from the AMI.
2. Run the following command:

```
sudo zypper install gcc
```

3. Install PHP 7.

```
sudo yum install php70
```

4. Download the Amazon ElastiCache Cluster Client.

```
wget https://elasticache-downloads.s3.amazonaws.com/ClusterClient/PHP-7.0/
latest-64bit
```

5. Extract `latest-64bit`.

```
tar -zxvf latest-64bit
```

6. With root permission, copy the extracted artifact file `amazon-elasticache-cluster-client.so` into `/usr/lib64/php7/extensions/`.

```
sudo mv artifact/amazon-elasticache-cluster-client.so /usr/lib64/php7/extensions/
```

7. Insert the line `extension=amazon-elasticache-cluster-client.so` into the file `/etc/php7/cli/php.ini`.

```
echo "extension=amazon-elasticache-cluster-client.so" | sudo tee --append /etc/
php7/cli/php.ini
```

8. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

### Installing PHP 5.x for new users

#### Topics

- To install PHP 5 on an Amazon Linux AMI 2014.03 (64-bit and 32-bit)
- To install PHP 5 on a Red Hat Enterprise Linux 7.0 AMI (64-bit and 32-bit)
- To install PHP 5 on a Ubuntu server 14.04 LTS AMI (64-bit and 32-bit)
- To install PHP 5 for SUSE Linux enterprise server 11 AMI (64-bit or 32-bit)
- Other Linux distributions

#### To install PHP 5 on an Amazon Linux AMI 2014.03 (64-bit and 32-bit)

1. Launch an Amazon Linux instance (either 64-bit or 32-bit) and log into it.
2. Install PHP dependencies:

```
sudo yum install gcc-c++ php php-pear
```

3. Download the correct `php-memcached` package for your Amazon EC2 instance and PHP version. For more information, see Downloading the installation package.
4. Install `php-memcached`. The URI should be the download path for the installation package:

```
sudo pecl install <package download path>
```

Here is a sample installation command for PHP 5.4, 64-bit Linux. In this sample, replace *X.Y.Z* with the actual version number:

```
sudo pecl install /home/AmazonElastiCacheClusterClient-X.Y.Z-PHP54-64bit.tgz
```

> **Note**
>
> Be sure to use the latest version of the install artifact.

5. With root/sudo permission, add a new file named `memcached.ini` in the `/etc/php.d` directory, and insert "extension=amazon-elasticache-cluster-client.so" in the file:

```
echo "extension=amazon-elasticache-cluster-client.so" | sudo tee --append /etc/
php.d/memcached.ini
```

6. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

**To install PHP 5 on a Red Hat Enterprise Linux 7.0 AMI (64-bit and 32-bit)**

1. Launch a Red Hat Enterprise Linux instance (either 64-bit or 32-bit) and log into it.
2. Install PHP dependencies:

```
sudo yum install gcc-c++ php php-pear
```

3. Download the correct `php-memcached` package for your Amazon EC2 instance and PHP version. For more information, see Downloading the installation package.
4. Install `php-memcached`. The URI should be the download path for the installation package:

```
sudo pecl install <package download path>
```

5. With root/sudo permission, add a new file named `memcached.ini` in the `/etc/php.d` directory, and insert `extension=amazon-elasticache-cluster-client.so` in the file.

```
echo "extension=amazon-elasticache-cluster-client.so" | sudo tee --append /etc/
php.d/memcached.ini
```

6. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

**To install PHP 5 on a Ubuntu server 14.04 LTS AMI (64-bit and 32-bit)**

1. Launch an Ubuntu Linux instance (either 64-bit or 32-bit) and log into it.
2. Install PHP dependencies:

```
sudo apt-get update
sudo apt-get install gcc g++ php5 php-pear
```

3. Download the correct `php-memcached` package for your Amazon EC2 instance and PHP version. For more information, see Downloading the installation package.
4. Install `php-memcached`. The URI should be the download path for the installation package.

```
sudo pecl install <package download path>
```

> **Note**
>
> This installation step installs the build artifact `amazon-elasticache-cluster-client.so` into the `/usr/lib/php5/20121212*` directory. Verify the absolute path of the build artifact, because you need it in the next step.

If the previous command doesn't work, you need to manually extract the PHP client artifact `amazon-elasticache-cluster-client.so` from the downloaded `*.tgz` file, and copy it to the `/usr/lib/php5/20121212*` directory.

```
tar -xvf <package download path>
cp amazon-elasticache-cluster-client.so /usr/lib/php5/20121212/
```

5. With root/sudo permission, add a new file named `memcached.ini` in the `/etc/php5/cli/conf.d` directory, and insert "extension=<absolute path to amazon-elasticache-cluster-client.so>" in the file.

```
echo "extension=<absolute path to amazon-elasticache-cluster-client.so>" | sudo tee
 --append /etc/php5/cli/conf.d/memcached.ini
```

6. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

**To install PHP 5 for SUSE Linux enterprise server 11 AMI (64-bit or 32-bit)**

1. Launch a SUSE Linux instance (either 64-bit or 32-bit) and log into it.
2. Install PHP dependencies:

```
sudo zypper install gcc php53-devel
```

3. Download the correct `php-memcached` package for your Amazon EC2 instance and PHP version. For more information, see Downloading the installation package.
4. Install `php-memcached`. The URI should be the download path for the installation package.

```
sudo pecl install <package download path>
```

5. With root/sudo permission, add a new file named `memcached.ini` in the `/etc/php5/conf.d` directory, and insert **`extension=amazon-elasticache-cluster-client.so`** in the file.

```
echo "extension=amazon-elasticache-cluster-client.so" | sudo tee --append /etc/
php5/conf.d/memcached.ini
```

6. Start or restart your Apache server.

```
sudo /etc/init.d/httpd start
```

**Note**

If Step 5 doesn't work for any of the previous platforms, verify the install path for `amazon-elasticache-cluster-client.so`. Also, specify the full path of the binary in the extension. In addition, verify that the PHP in use is a supported version. We support versions 5.3 through 5.5.

### Other Linux distributions

On some systems, notably CentOS7 and Red Hat Enterprise Linux (RHEL) 7.1, `libsasl2.so.3` has replaced `libsasl2.so.2`. On those systems, when you load the ElastiCache cluster client, it attempts and fails to find and load `libsasl2.so.2`. To resolve this issue, create a symbolic link to `libsasl2.so.3` so that when the client attempts to load libsasl2.so.2, it is redirected to `libsasl2.so.3`. The following code creates this symbolic link.

```
cd /usr/lib64
sudo ln libsasl2.so.3 libsasl2.so.2
```

### Removing the PHP cluster client

#### Topics

- Removing an earlier version of PHP 7
- Removing an earlier version of PHP 5

### Removing an earlier version of PHP 7

#### To remove an earlier version of PHP 7

1. Remove the `amazon-elasticache-cluster-client.so` file from the appropriate PHP lib directory as previously indicated in the installation instructions. See the section for your installation at For users who already have *php-memcached* extension installed.
2. Remove the line `extension=amazon-elasticache-cluster-client.so` from the `php.ini` file.
3. Start or restart your Apache server.

   ```
   sudo /etc/init.d/httpd start
   ```

### Removing an earlier version of PHP 5

#### To remove an earlier version of PHP 5

1. Remove the `php-memcached` extension:

```
sudo pecl uninstall __uri/AmazonElastiCacheClusterClient
```

2. Remove the `memcached.ini` file added in the appropriate directory as indicated in the previous installation steps.

### Compiling the source code for the ElastiCache cluster client for PHP

This section covers how to obtain and compile the source code for the ElastiCache Cluster Client for PHP.

There are two packages you need to pull from GitHub and compile; aws-elasticache-cluster-client-libmemcached and aws-elasticache-cluster-client-memcached-for-php.

**Topics**

- Compiling the libmemcached library
- Compiling the ElastiCache Memcached auto discovery client for PHP

### Compiling the libmemcached library

**To compile the aws-elasticache-cluster-client-libmemcached library**

1. Launch an Amazon EC2 instance.
2. Install the library dependencies.
   - On Amazon Linux 201509 AMI

     ```
     sudo yum install gcc gcc-c++ autoconf libevent-devel
     ```

   - On Ubuntu 14.04 AMI

     ```
     sudo apt-get update
     sudo apt-get install libevent-dev gcc g++ make autoconf libsasl2-dev
     ```

3. Pull the repository and compile the code.

   ```
   Download and install  https://github.com/awslabs/aws-elasticache-cluster-client-libmemcached/archive/v1.0.18.tar.gz
   ```

### Compiling the ElastiCache Memcached auto discovery client for PHP

The following sections describe how to compile the ElastiCache Memcached Auto Discovery Client

**Topics**

- Compiling the ElastiCache Memcached client for PHP 7

- Compiling the ElastiCache Memcached client for PHP 5

### Compiling the ElastiCache Memcached client for PHP 7

Run the following set of commands under the code directory. In these commands, replace *<libmemcached-install-directory>* with the path where you installed the aws-elasticache-cluster-client-libmemcached library in the previous step. If you compiled and installed with the default prefix, this path is typically /usr/local.

```
git clone https://github.com/awslabs/aws-elasticache-cluster-client-memcached-for-
php.git
cd aws-elasticache-cluster-client-memcached-for-php
git checkout php7
sudo yum install php70-devel
phpize
./configure --with-libmemcached-dir=<libmemcached-install-directory> --disable-
memcached-sasl
make
make install
```

**Note**

You can statically link the libmemcached library into the PHP binary so it can be ported across various Linux platforms. To do that, run the following command before make:

```
sed -i "s#-lmemcached#<libmemcached-install-directory>/lib/libmemcached.a -
lcrypt -lpthread -lm -lstdc++ -lsasl2#" Makefile
```

### Compiling the ElastiCache Memcached client for PHP 5

Compile the aws-elasticache-cluster-client-memcached-for-php by running the following commands under the aws-elasticache-cluster-client-memcached-for-php/ folder. In these commands, replace *<libmemcached-install-directory>* with the path where you installed the aws-elasticache-cluster-client-libmemcached library. If you compiled and installed with the default prefix, this path is typically /usr/local.

```
git clone https://github.com/awslabs/aws-elasticache-cluster-client-memcached-for-
php.git
```

```
cd aws-elasticache-cluster-client-memcached-for-php
sudo yum install zlib-devel
phpize
./configure --with-libmemcached-dir=<libmemcached-install-directory>
make
make install
```

## Configuring ElastiCache clients

An ElastiCache cluster is protocol-compliant with Valkey, Memcached, and Redis OSS. The code, applications, and most popular tools that you use today with your existing environment will work seamlessly with the service.

This section discusses specific considerations for connecting to cache nodes in ElastiCache.

### Topics


- Restricted commands
- Finding node endpoints and port numbers
- Connecting for using auto discovery
- Connecting to nodes in a Valkey or Redis OSS cluster
- DNS names and underlying IP


### Restricted commands

To deliver a managed service experience, ElastiCache restricts access to certain cache engine-specific commands that require advanced privileges. For clusters running Redis OSS, the following commands are unavailable:

- `bgrewriteaof`
- `bgsave`
- `config`
- `debug`
- `migrate`
- `replicaof`
- `save`
- `slaveof`
- `shutdown`
- `sync`

### Finding node endpoints and port numbers

To connect to a cache node, your application needs to know the endpoint and port number for that node.

### Finding node endpoints and port numbers (Console)

#### To determine node endpoints and port numbers

1. Sign in to the Amazon ElastiCache management console and choose the engine running on your cluster.

   A list of all clusters running the chosen engine appears.
2. Continue below for the engine and configuration you're running.
3. Choose the name of the cluster of interest.
4. Locate the **Port** and **Endpoint** columns for the node you're interested in.

### Finding cache node endpoints and port numbers (AWS CLI)

To determine cache node endpoints and port numbers, use the command `describe-cache-clusters` with the `--show-cache-node-info` parameter.

```
aws elasticache describe-cache-clusters --show-cache-node-info
```

The fully qualified DNS names and port numbers are in the Endpoint section of the output.

### Finding cache node endpoints and port numbers (ElastiCache API)

To determine cache node endpoints and port numbers, use the action `DescribeCacheClusters` with the `ShowCacheNodeInfo=true` parameter.

#### Example

```
https://elasticache.us-west-2.amazonaws.com /
    ?Action=DescribeCacheClusters
    &ShowCacheNodeInfo=true
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
```

```
    &Timestamp=20140421T220302Z
    &Version=2014-09-30
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Credential=<credential>
    &X-Amz-Date=20140421T220302Z
    &X-Amz-Expires=20140421T220302Z
    &X-Amz-Signature=<signature>
    &X-Amz-SignedHeaders=Host
```

### Connecting for using auto discovery

If your applications use Auto Discovery, you only need to know the configuration endpoint for the cluster, rather than the individual endpoints for each cache node. For more information, see Automatically identify nodes in your cluster (Memcached).

> **Note**
>
> At this time, Auto Discovery is only available for clusters running Memcached.

### Connecting to nodes in a Valkey or Redis OSS cluster

> **Note**
>
> At this time, clusters (API/CLI: replication groups) that support replication and read replicas are only supported for clusters running Valkey or Redis OSS.

For clusters, ElastiCache provides console, CLI, and API interfaces to obtain connection information for individual nodes.

For read-only activity, applications can connect to any node in the cluster. However, for write activity, we recommend that your applications connect to the primary endpoint (Valkey or Redis OSS (cluster mode disabled)) or configuration endpoint (Valkey or Redis OSS (cluster mode enabled)) for the cluster instead of connecting directly to a node. This will ensure that your applications can always find the correct node, even if you decide to reconfigure your cluster by promoting a read replica to the primary role.

### Connecting to clusters in a replication group (Console)

#### To determine endpoints and port numbers

- See the topic, Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console).

### Connecting to clusters in a replication group (AWS CLI)

#### To determine cache node endpoints and port numbers

Use the command `describe-replication-groups` with the name of your replication group:

```
aws elasticache describe-replication-groups redis2x2
```

This command should produce output similar to the following:

```
{
    "ReplicationGroups": [
        {
            "Status": "available",
            "Description": "2 shards, 2 nodes (1 + 1 replica)",
            "NodeGroups": [
                {
                    "Status": "available",
                    "Slots": "0-8191",
                    "NodeGroupId": "0001",
                    "NodeGroupMembers": [
                        {
                            "PreferredAvailabilityZone": "us-west-2c",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "redis2x2-0001-001"
                        },
                        {
                            "PreferredAvailabilityZone": "us-west-2a",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "redis2x2-0001-002"
                        }
                    ]
                },
                {
                    "Status": "available",
```

```
                    "Slots": "8192-16383",
                    "NodeGroupId": "0002",
                    "NodeGroupMembers": [
                        {
                            "PreferredAvailabilityZone": "us-west-2b",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "redis2x2-0002-001"
                        },
                        {
                            "PreferredAvailabilityZone": "us-west-2a",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "redis2x2-0002-002"
                        }
                    ]
                }
            ],
            "ConfigurationEndpoint": {
                "Port": 6379,
                "Address": "redis2x2.9dcv5r.clustercfg.usw2.cache.amazonaws.com"
            },
            "ClusterEnabled": true,
            "ReplicationGroupId": "redis2x2",
            "SnapshotRetentionLimit": 1,
            "AutomaticFailover": "enabled",
            "SnapshotWindow": "13:00-14:00",
            "MemberClusters": [
                "redis2x2-0001-001",
                "redis2x2-0001-002",
                "redis2x2-0002-001",
                "redis2x2-0002-002"
            ],
            "CacheNodeType": "cache.m5.large",
            "PendingModifiedValues": {}
        }
    ]
}
```

### Connecting to clusters in a replication group (ElastiCache API)

#### To determine cache node endpoints and port numbers

Call `DescribeReplicationGroups` with the following parameter:

`ReplicationGroupId` = the name of your replication group.

### Example

```
https://elasticache.us-west-2.amazonaws.com /
    ?Action=DescribeCacheClusters
    &ReplicationGroupId=repgroup01
    &Version=2014-09-30
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20140421T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20140421T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20140421T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

### DNS names and underlying IP

Clients maintain a server list containing the addresses and ports of the servers holding the cache data. When using ElastiCache, the DescribeCacheClusters API (or the describe-cache-clusters command line utility) returns a fully qualified DNS entry and port number that can be used for the server list.

> **Important**
>
> It is important that client applications are configured to frequently resolve DNS names of cache nodes when they attempt to connect to a cache node endpoint.

ElastiCache ensures that the DNS name of cache nodes remain the same when cache nodes are recovered in case of failure.

Most client libraries support persistent cache node connections by default. We recommend using persistent cache node connections when using ElastiCache. Client-side DNS caching can occur in multiple places, including client libraries, the language runtime, or the client operating system. You should review your application configuration at each layer to ensure that you are frequently resolving IP addresses for your cache nodes.

# Data tiering in ElastiCache

ElastiCache for Valkey or Redis OSS clusters that comprise a replication group and use a node type from the r6gd, r7gd, or r8gd families have their data tiered between memory and local SSD (solid state drives) storage. Data tiering provides a new price-performance option for Valkey or Redis OSS workloads by utilizing lower-cost solid state drives (SSDs) in each cluster node in addition to storing data in memory. It is ideal for workloads that access up to 20 percent of their overall dataset regularly, and for applications that can tolerate additional latency when accessing data on SSD.

On ElastiCache clusters with data tiering, ElastiCache monitors the last access time of every item it stores. When available memory (DRAM) is fully consumed, ElastiCache uses a least-recently used (LRU) algorithm to automatically move infrequently accessed items from memory to SSD. When data on SSD is subsequently accessed, ElastiCache automatically and asynchronously moves it back to memory before processing the request. If you have a workload that accesses only a subset of its data regularly, data tiering is an optimal way to scale your capacity cost-effectively.

Note that when using data tiering, keys themselves always remain in memory, while the LRU governs the placement of values on memory vs. disk. In general, we recommend that your key sizes are smaller than your value sizes when using data tiering.

Data tiering is designed to have minimal performance impact to application workloads. For example, assuming 500-byte String values, you can expect an additional 300 microseconds of latency on average for requests to data stored on SSD compared to requests to data in memory.

With the largest data tiering node size (cache.r6gd.16xlarge), you can store up to 1 petabyte in a single 500-node cluster (500 TB when using 1 read replica). Data tiering is compatible with all Valkey or Redis OSS commands and data structures supported in ElastiCache. You don't need any client-side changes to use this feature.

**Topics**


- Best practices
- Limitations
- Pricing
- Monitoring
- Using data tiering
- Restoring data from backup into clusters with data tiering enabled

## Best practices

We recommend the following best practices:

- Data tiering is ideal for workloads that access up to 20 percent of their overall dataset regularly, and for applications that can tolerate additional latency when accessing data on SSD.
- When using SSD capacity available on data-tiered nodes, we recommend that value size be larger than the key size. When items are moved between DRAM and SSD, keys will always remain in memory and only the values are moved to the SSD tier.

## Limitations

Data tiering has the following limitations:

- You can only use data tiering on clusters that are part of a replication group.
- The node type you use must be from the r6gd, r7gd, or r8gd families.
- For the r7gd and r8gd families, you must use Valkey 7.2 or later. For the r6gd family, you must use Valkey 7.2 or later, or Redis OSS 6.2 or later.
- You can't restore a backup from a data tiering cluster into a cluster that doesn't use data tiering.
- You cannot export a backup to Amazon S3 for data-tiering clusters.
- Online migration is not supported for data tiering node types.
- Scaling between clusters with data tiering enabled and data tiering disabled is not supported. To migrate data from an ElastiCache cluster with data tiering disabled to a cluster with data tiering enabled, you can restore a backup to a new cluster with data tiering enabled. For more information, see Scaling ElastiCache.
- Auto scaling is supported on clusters using data tiering for Valkey version 7.2 and later, and Redis OSS version 7.0.7 and later. For more information, see Auto Scaling Valkey and Redis OSS clusters
- Data tiering only supports `volatile-lru`, `allkeys-lru`, `volatile-lfu`, `allkeys-lfu` and `noeviction` maxmemory policies.
- Forkless save is supported for Valkey version 7.2 and later, and Redis OSS version 7.0.7 and later.
- Items larger than 128 MiB are not moved to SSD.
- For Valkey 8.1 and later, items with a combined key and value size of less than 40 bytes are not moved to SSD.
- For Valkey 9.0 and later, hash items with field-level TTLs are not moved to SSD.

- Data tiering is not supported with durability-enabled clusters.

## Pricing

Data tiering offers a lower-cost way to scale your cache capacity by using local SSD storage in each node in addition to memory. R6gd nodes have 4.8x more total capacity (memory + SSD) and can help you achieve over 60 percent savings when running at maximum utilization compared to R6g nodes (memory only). For more information, see ElastiCache pricing.

## Monitoring

ElastiCache offers metrics designed specifically to monitor the performance clusters that use data tiering. To monitor the ratio of items in DRAM compared to SSD, you can use the `CurrItems` metric at Metrics for Valkey and Redis OSS. You can calculate the percentage as: *(CurrItems with Dimension: Tier = Memory * 100) / (CurrItems with no dimension filter)*.

If the configured eviction policy allows, then ElastiCache will start evicting items when less than 5 percent of available memory (DRAM) remains. On nodes configured with noeviction policy, write operations will receive an out of memory error.

It is still recommended that you consider scaling out for Cluster Mode Enabled clusters or scaling up for Cluster Mode disabled clusters when less than 5 percent of available memory (DRAM) remains. For more information on scaling see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters. For more information on metrics for Valkey or Redis OSS clusters that use data tiering see Metrics for Valkey and Redis OSS.

## Using data tiering

### Using data tiering using the AWS Management Console

When creating a cluster as part of a replication group, you use data tiering by selecting a node type from the r6gd, r7gd, or r8gd families, such as *cache.r6gd.xlarge*. Selecting that node type automatically enables data tiering.

For more information on creating a cluster, see Creating a cluster for Valkey or Redis OSS.

### Enabling data tiering using the AWS CLI

When creating a replication group using the AWS CLI, you use data tiering by selecting a node type from the r6gd, r7gd, or r8gd families, such as *cache.r6gd.xlarge* and setting the `--data-tiering-enabled` parameter.

Data tiering node types always use data tiering. If you set the `--no-data-tiering-enabled` parameter, the operation fails.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
   --replication-group-id redis-dt-cluster \
   --replication-group-description "Redis OSS cluster with data tiering" \
   --num-node-groups 1 \
   --replicas-per-node-group 1 \
   --cache-node-type cache.r6gd.xlarge \
   --engine redis \
   --cache-subnet-group-name default \
   --automatic-failover-enabled \
   --data-tiering-enabled
```

For Windows:

```
aws elasticache create-replication-group ^
   --replication-group-id redis-dt-cluster ^
   --replication-group-description "Redis OSS cluster with data tiering" ^
   --num-node-groups 1 ^
   --replicas-per-node-group 1 ^
   --cache-node-type cache.r6gd.xlarge ^
   --engine redis ^
   --cache-subnet-group-name default ^
   --automatic-failover-enabled ^
   --data-tiering-enabled
```

After running this operation, you will see a response similar to the following:

```
{
    "ReplicationGroup": {
        "ReplicationGroupId": "redis-dt-cluster",
        "Description": "Redis OSS cluster with data tiering",
        "Status": "creating",
        "PendingModifiedValues": {},
        "MemberClusters": [
            "redis-dt-cluster"
        ],
        "AutomaticFailover": "enabled",
        "DataTiering": "enabled",
```

```
        "SnapshotRetentionLimit": 0,
        "SnapshotWindow": "06:00-07:00",
        "ClusterEnabled": false,
        "CacheNodeType": "cache.r6gd.xlarge",
        "TransitEncryptionEnabled": false,
        "AtRestEncryptionEnabled": false
    }
}
```

## Restoring data from backup into clusters with data tiering enabled

You can restore a backup to a new cluster with data tiering enabled using the (Console), (AWS CLI) or (ElastiCache API). When you create a cluster using a node type from the r6gd, r7gd, or r8gd families, data tiering is enabled.

### Restoring data from backup into clusters with data tiering enabled (console)

**To restore a backup to a new cluster with data tiering enabled (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Backups**.
3. In the list of backups, choose the box to the left of the backup name you want to restore from.
4. Choose **Restore**.
5. Complete the **Restore Cluster** dialog box. Be sure to complete all the **Required** fields and any of the others you want to change from the defaults.

    1. **Cluster ID** – Required. The name of the new cluster.
    2. **Cluster mode enabled (scale out)** – Choose this for a Valkey or Redis OSS (cluster mode enabled) cluster.
    3. **Node Type** – Specify **cache.r6gd.xlarge** or any other node type from the r6gd, r7gd, or r8gd families.
    4. **Number of Shards** – Choose the number of shards you want in the new cluster (API/CLI: node groups).
    5. **Replicas per Shard** – Choose the number of read replica nodes you want in each shard.
    6. **Slots and keyspaces** – Choose how you want keys distributed among the shards. If you choose to specify the key distributions complete the table specifying the key ranges for each shard.

7. **Availability zone(s)** – Specify how you want the cluster's Availability Zones selected.
8. **Port** – Change this only if you want the new cluster to use a different port.
9. **Choose a VPC** – Choose the VPC in which to create this cluster.
10. **Parameter Group** – Choose a parameter group that reserves sufficient memory for Valkey or Redis OSS overhead for the node type you selected.

6. When the settings are as you want them, choose **Create**.

For more information on creating a cluster, see Creating a cluster for Valkey or Redis OSS.

**Restoring data from backup into clusters with data tiering enabled (AWS CLI)**

When creating a replication group using the AWS CLI, data tiering is by default used by selecting a node type from the r6gd, r7gd, or r8gd families, such as *cache.r6gd.xlarge* and setting the `--data-tiering-enabled` parameter.

Data tiering node types always use data tiering. If you set the `--no-data-tiering-enabled` parameter, the operation fails.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
   --replication-group-id redis-dt-cluster \
   --replication-group-description "Redis OSS cluster with data tiering" \
   --num-node-groups 1 \
   --replicas-per-node-group 1 \
   --cache-node-type cache.r6gd.xlarge \
   --engine redis \
   --cache-subnet-group-name default \
   --automatic-failover-enabled \
   --data-tiering-enabled \
   --snapshot-name my-snapshot
```

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group ^
   --replication-group-id redis-dt-cluster ^
   --replication-group-description "Redis OSS cluster with data tiering" ^
   --num-node-groups 1 ^
   --replicas-per-node-group 1 ^
   --cache-node-type cache.r6gd.xlarge ^
```

```
    --engine redis ^
    --cache-subnet-group-name default ^
    --automatic-failover-enabled ^
    --data-tiering-enabled ^
    --snapshot-name my-snapshot
```

After running this operation, you will see a response similar to the following:

```
{
    "ReplicationGroup": {
        "ReplicationGroupId": "redis-dt-cluster",
        "Description": "Redis OSS cluster with data tiering",
        "Status": "creating",
        "PendingModifiedValues": {},
        "MemberClusters": [
            "redis-dt-cluster"
        ],
        "AutomaticFailover": "enabled",
        "DataTiering": "enabled",
        "SnapshotRetentionLimit": 0,
        "SnapshotWindow": "06:00-07:00",
        "ClusterEnabled": false,
        "CacheNodeType": "cache.r6gd.xlarge",
        "TransitEncryptionEnabled": false,
        "AtRestEncryptionEnabled": false
    }
}
```

## Preparing a cluster in ElastiCache

Following, you can find instructions on creating a cluster using the ElastiCache console, the AWS CLI, or the ElastiCache API.

You can also create an ElastiCache cluster using AWS CloudFormation. For more information, see AWS::ElastiCache::CacheCluster in the *AWS Cloud Formation User Guide*, which includes guidance on how to implement that approach.

Whenever you create a cluster or replication group, it is a good idea to do some preparatory work so you won't need to upgrade or make changes right away.

### Topics

- Determining your ElastiCache cluster requirements

- Choosing your node size

## Determining your ElastiCache cluster requirements

### Preparation

Knowing the answers to the following questions helps make creating your ElastiCache cluster go smoother:

- Which node instance type do you need?

  For guidance on choosing an instance node type, see Choosing your node size.

- Will you launch your cluster in a virtual private cloud (VPC) based on Amazon VPC?

  > **Important**
  >
  > If you're going to launch your cluster in a VPC, make sure to create a subnet group in the same VPC before you start creating a cluster. For more information, see Subnets and subnet groups.
  > ElastiCache is designed to be accessed from within AWS using Amazon EC2. However, if you launch in a VPC based on Amazon VPC and your cluster is in an VPC, you can provide access from outside AWS. For more information, see Accessing ElastiCache resources from outside AWS.

- Do you need to customize any parameter values?

  If you do, create a custom parameter group. For more information, see Creating an ElastiCache parameter group.

  If you're running Valkey or Redis OSS, consider setting `reserved-memory` or `reserved-memory-percent`. For more information, see Managing reserved memory for Valkey and Redis OSS.

- Do you need to create your own *VPC security group*?

  For more information, see Security in Your VPC.

- How do you intend to implement fault tolerance?

  For more information, see Mitigating Failures.

**Topics**


- ElastiCache memory and processor requirements
- Memcached cluster configuration
- Valkey and Redis OSS cluster configuration
- ElastiCache scaling requirements
- ElastiCache access requirements
- Region, Availability Zone and Local Zone requirements for ElastiCache
- Do you need durability?

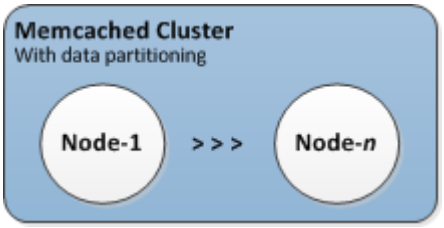

### ElastiCache memory and processor requirements

The basic building block of Amazon ElastiCache is the node. Nodes are configured singularly or in groupings to form clusters. When determining the node type to use for your cluster, take the cluster's node configuration and the amount of data you have to store into consideration.

The Memcached engine is multi-threaded, so a node's number of cores impacts the compute power available to the cluster.

### Memcached cluster configuration

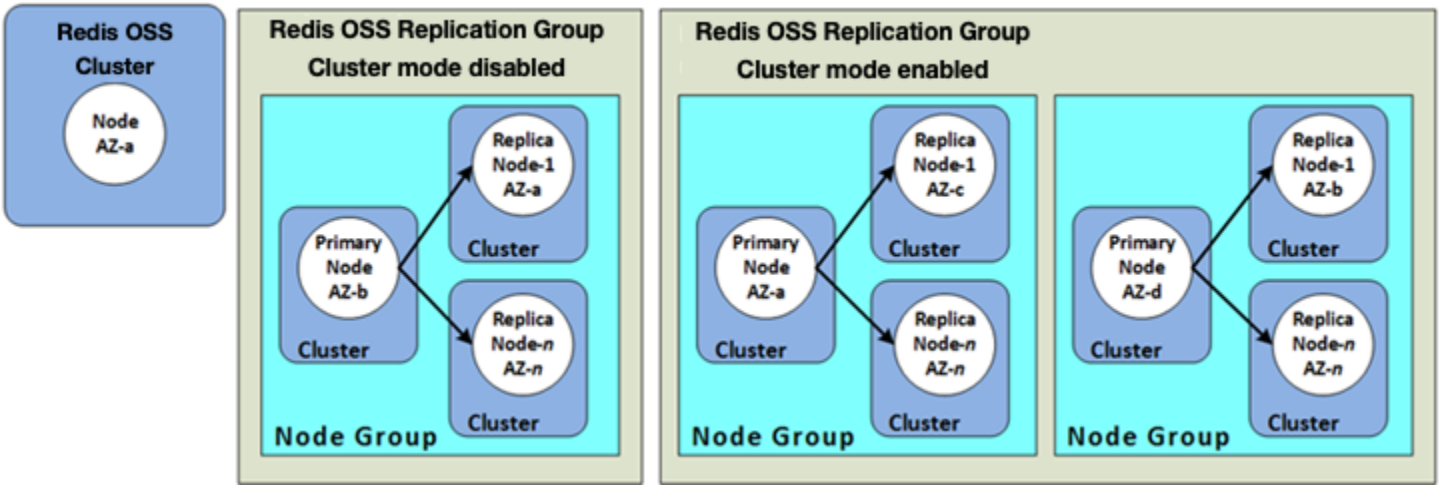
ElastiCache for Memcached clusters are comprised of from 1 to 60 nodes. The data in a Memcached cluster is partitioned across the nodes in the cluster. Your application connects with a Memcached cluster using a network address called an Endpoint. Each node in a Memcached cluster has its own endpoint which your application uses to read from or write to the specific node. In addition to the node endpoints, the Memcached cluster itself has an endpoint called the *configuration endpoint*. Your application can use this endpoint to read from or write to the cluster, leaving the determination of which node to read from or write to up to auto discovery.




For more information, see Managing clusters in ElastiCache.

### Valkey and Redis OSS cluster configuration

ElastiCache for Valkey and Redis OSS clusters are comprised of from 0 to 500 shards (also called node groups). The data in a Valkey or Redis OSS cluster is partitioned across the shards in the cluster. Your application connects with a Valkey or Redis OSS cluster using a network address called an Endpoint. The nodes in a Valkey or Redis OSS shard fulfill one of two roles: one read/write primary and all other nodes read-only secondaries (also called read replicas). In addition to the node endpoints, the Valkey or Redis OSS cluster itself has an endpoint called the *configuration endpoint*. Your application can use this endpoint to read from or write to the cluster, leaving the determination of which node to read from or write to up to ElastiCache for Redis OSS.




For more information, see Managing clusters in ElastiCache.

### ElastiCache scaling requirements

All clusters can be scaled up by creating a new cluster with the new, larger node type. When you scale up a Memcached cluster, the new cluster starts out empty. When you scale up a Valkey or Redis OSS cluster, you can seed it from a backup and avoid having the new cluster start out empty.

Amazon ElastiCache for Memcached clusters can be scaled out or in. To scale a Memcached cluster out or in you merely add or remove nodes from the cluster. If you have enabled Automatic Discovery and your application is connecting to the cluster's configuration endpoint, you do not need to make any changes in your application when you add or remove nodes.

For more information, see Scaling ElastiCache in this guide.

### ElastiCache access requirements

By design, Amazon ElastiCache clusters are accessed from Amazon EC2 instances. Network access to an ElastiCache cluster is limited to the account that created the cluster. Therefore, before you

can access a cluster from an Amazon EC2 instance, you must authorize the Amazon EC2 instance to access the cluster. The steps to do this vary, depending upon whether you launched into EC2-VPC or EC2-Classic.

If you launched your cluster into EC2-VPC you need to grant network ingress to the cluster. If you launched your cluster into EC2-Classic you need to grant the Amazon Elastic Compute Cloud security group associated with the instance access to your ElastiCache security group. For detailed instructions, see Step 3. Authorize access to the cluster in this guide.

**Region, Availability Zone and Local Zone requirements for ElastiCache**

Amazon ElastiCache supports all AWS regions. By locating your ElastiCache clusters in an AWS Region close to your application you can reduce latency. If your cluster has multiple nodes, locating your nodes in different Availability Zones or in Local Zones can reduce the impact of failures on your cluster.

For more information, see the following:

- Choosing regions and availability zones for ElastiCache
- Using local zones with ElastiCache
- Mitigating Failures

**Do you need durability?**

Do you need durability for your data? If your application requires durability beyond in-memory replication, you can create a Valkey 9.0+ cluster with durability enabled. Durability is set at cluster creation time and cannot be enabled or disabled afterward. For more information, see the section called “Durability”.

## Choosing your node size

The node size you select for your ElastiCache cluster impacts costs, performance, and fault tolerance.

**Node size (Valkey and Redis OSS)**

For information about the benefits of Graviton processors, see AWS Graviton Processor.

Answering the following questions can help you determine the minimum node type you need for your Valkey or Redis OSS implementation:

- Do you expect throughput-bound workloads with multiple client connections?

  If this is the case and you're running Valkey, or Redis OSS version 5.0.6 to 7.1, you can get better throughput and latency with the enhanced I/O feature. Available CPUs offload the client connections from the engine. If you're running Valkey, or Redis OSS version 7.0.4 to 7.1, on top of enhanced I/O, you get additional acceleration with enhanced I/O multiplexing, where each dedicated network I/O thread pipelines commands from multiple clients into the engine, taking advantage of its ability to efficiently process commands in batches. Starting with ElastiCache for Redis OSS v7.1 and Valkey 7.2, the enhanced I/O threads functionality also handles the presentation layer logic. This means that enhanced I/O threads not only read client input, but also parse the input into the engine's binary command format, which is then forwarded to the main thread for execution, providing performance gain. For more information, see the blog post and the supported versions page.

- Do you have workloads that access a small percentage of their data regularly?

  If this is the case, you can leverage data tiering by choosing a data tiering node type. The r7gd and r8gd families require Valkey 7.2 or later. The r6gd family requires Valkey 7.2 or later, or Redis OSS 6.2 or later. With data tiering, least-recently used data is stored in SSD. When it is retrieved there is a small latency cost, which is balanced by cost savings. For more information, see Data tiering in ElastiCache.

  For more information, see Supported node types.

- How much total memory do you need for your data?

  To get a general estimate, take the size of the items that you want to cache. Multiply this size by the number of items that you want to keep in the cache at the same time. To get a reasonable estimation of the item size, first serialize your cache items, then count the characters. Then divide this over the number of shards in your cluster.

  For more information, see Supported node types.

- Valkey and Redis OSS use a forkless save process for failover, snapshot, synchronizing, and promoting a replica to primary operations. This process requires less available memory because writes are managed without forking the process.

  For more information, see the following:

  - Ensuring you have enough memory to make a Valkey or Redis OSS snapshot

- How write-heavy is your application?

Write heavy applications can require significantly more available memory, memory not used by data, when taking snapshots or failing over. Whenever the BGSAVE process is performed, you must have sufficient memory that is unused by data to accommodate all the writes that transpire during the BGSAVE process. Examples are when taking a snapshot, when syncing a primary cluster with a replica in a cluster, and when enabling the append-only file (AOF) feature. Another is when promoting a replica to primary (if you have Multi-AZ enabled). The worst case is when all of your data is rewritten during the process. In this case, you need a node instance size with twice as much memory as needed for data alone.

For more detailed information, see Ensuring you have enough memory to make a Valkey or Redis OSS snapshot.

- Will your implementation be a standalone Valkey or Redis OSS (cluster mode disabled) cluster, or a Valkey or Redis OSS (cluster mode enabled) cluster with multiple shards?

  **Valkey or Redis OSS (cluster mode disabled) cluster**

  If you're implementing a Valkey or Redis OSS (cluster mode disabled) cluster, your node type must be able to accommodate all your data plus the necessary overhead as described in the previous bullet.

  For example, suppose that you estimate that the total size of all your items is 12 GB. In this case, you can use a `cache.m5.xlarge` node with 12.93 GB of memory or a `cache.r5.large` node with 13.07 GB of memory. However, you might need more memory for BGSAVE operations. If your application is write-heavy, double the memory requirements to at least 24 GB. Thus, use either a `cache.m5.2xlarge` with 26.04 GB of memory or a `cache.r5.xlarge` with 26.32 GB of memory.

  **Valkey or Redis OSS (cluster mode enabled) with multiple shards**

  If you're implementing a Valkey or Redis OSS (cluster mode enabled) cluster with multiple shards, then the node type must be able to accommodate `bytes-for-data-and-overhead / number-of-shards` bytes of data.

  For example, suppose that you estimate that the total size of all your items to be 12 GB and you have two shards. In this case, you can use a `cache.m5.large` node with 6.38 GB of memory (12 GB / 2). However, you might need more memory for BGSAVE operations. If your application is write-heavy, double the memory requirements to at least 12 GB per shard. Thus, use either a `cache.m5.xlarge` with 12.93 GB of memory or a `cache.r5.large` with 13.07 GB of memory.

- Are you using Local Zones?

  Local Zones enable you to place resources such as an ElastiCache cluster in multiple locations close to your users. But when you choose your node size, be aware that the available node sizes are limited to the following at this time, regardless of capacity requirements:

  - Current generation:

    **M5 node types:** `cache.m5.large`, `cache.m5.xlarge`, `cache.m5.2xlarge`, `cache.m5.4xlarge`, `cache.m5.12xlarge`, `cache.m5.24xlarge`

    **R5 node types:** `cache.r5.large`, `cache.r5.xlarge`, `cache.r5.2xlarge`, `cache.r5.4xlarge`, `cache.r5.12xlarge`, `cache.r5.24xlarge`

    **T3 node types:** `cache.t3.micro`, `cache.t3.small`, `cache.t3.medium`

While your cluster is running, you can monitor the memory usage, processor utilization, cache hits, and cache misses metrics that are published to CloudWatch. You might notice that your cluster doesn't have the hit rate that you want or that keys are being evicted too often. In these cases, you can choose a different node size with larger CPU and memory specifications.

When monitoring CPU usage, remember Valkey and Redis OSS are single-threaded. Thus, multiply the reported CPU usage by the number of CPU cores to get that actual usage. For example, a four-core CPU reporting a 20-percent usage rate is actually the one core Redis OSS is running at 80 percent utilization.

**Node size (Memcached)**

Memcached clusters contain one or more nodes with the cluster's data partitioned across the nodes. Because of this, the memory needs of the cluster and the memory of a node are related, but not the same. You can attain your needed cluster memory capacity by having a few large nodes or several smaller nodes. Further, as your needs change, you can add nodes to or remove nodes from the cluster and thus pay only for what you need.

The total memory capacity of your cluster is calculated by multiplying the number of nodes in the cluster by the RAM capacity of each node after deducting system overhead. The capacity of each node is based on the node type.

```
cluster_capacity = number_of_nodes * (node_capacity - system_overhead)
```

The number of nodes in the cluster is a key factor in the availability of your cluster running Memcached. The failure of a single node can have an impact on the availability of your application and the load on your backend database. In such a case, ElastiCache provisions a replacement for a failed node and it gets repopulated. To reduce this availability impact, spread your memory and compute capacity over more nodes with smaller capacity, rather than using fewer high-capacity nodes.

In a scenario where you want to have 35 GB of cache memory, you can set up any of the following configurations:

- 11 `cache.t2.medium` nodes with 3.22 GB of memory and 2 threads each = 35.42 GB and 22 threads.
- 6 `cache.m4.large` nodes with 6.42 GB of memory and 2 threads each = 38.52 GB and 12 threads.
- 3 `cache.r4.large` nodes with 12.3 GB of memory and 2 threads each = 36.90 GB and 6 threads.
- 3 `cache.m4.xlarge` nodes with 14.28 GB of memory and 4 threads each = 42.84 GB and 12 threads.

**Comparing node options**

| Node type | Memory (in GiB) | Cores | Hourly cost * | Nodes needed | Total memory (in GiB) | Total cores | Monthly cost |
|---|---|---|---|---|---|---|---|
| cache.t2.medium | 3.22 | 2 | $ 0.068 | 11 | 35.42 | 22 | $ 538.56 |
| cache.m4.large | 6.42 | 2 | $ 0.156 | 6 | 38.52 | 12 | $ 673.92 |
| cache.m4.xlarge | 14.28 | 4 | $ 0.311 | 3 | 42.84 | 12 | $ 671.76 |
| cache.m5.xlarge | 12.93 | 4 | $ 0.311 | 3 | 38.81 | 12 | $ 671.76 |

| Node type | Memory (in GiB) | Cores | Hourly cost * | Nodes needed | Total memory (in GiB) | Total cores | Monthly cost |
|---|---|---|---|---|---|---|---|
| cache.m6g.large | 6.85 | 2 | $ 0.147 | 6 | 41.1 | 12 | $ 635 |
| cache.r4.large | 12.3 | 2 | $ 0.228 | 3 | 36.9 | 6 | $ 492.48 |
| cache.r5.large | 13.07 | 2 | $ 0.216 | 3 | 39.22 | 6 | $ 466.56 |
| cache.r6g.large | 13.07 | 2 | $ 0.205 | 3 | 42.12 | 6 | $ 442 |

* Hourly cost per node as of October 8, 2020.

Monthly cost at 100% usage for 30 days (720 hours).

These options each provide similar memory capacity but different computational capacity and cost. To compare the costs of your specific options, see Amazon ElastiCache Pricing.

For clusters running Memcached, some of the available memory on each node is used for connection overhead. For more information, see Memcached connection overhead

Using multiple nodes requires spreading the keys across them. Each node has its own endpoint. For easy endpoint management, you can use the ElastiCache the Auto Discovery feature, which enables client programs to automatically identify all of the nodes in a cluster. For more information, see Automatically identify nodes in your cluster (Memcached).

In some cases, you might be unsure how much capacity you need. If so, for testing we recommend starting with one `cache.m5.large` node. Then monitor the memory usage, CPU utilization, and cache hit rate with the ElastiCache metrics that are published to Amazon CloudWatch. For more information on CloudWatch metrics for ElastiCache, see Monitoring use with CloudWatch Metrics. For production and larger workloads, the R5 nodes provide the best performance and RAM cost value.

If your cluster doesn't have the hit rate that you want, you can easily add more nodes to increase the total available memory in your cluster.

If your cluster is bound by CPU but has sufficient hit rate, set up a new cluster with a node type that provides more compute power.

# Creating a cluster for Valkey or Redis OSS

The following examples show how to create a Valkey or Redis OSS cluster using the AWS Management Console, AWS CLI and ElastiCache API.

### Creating a Valkey or Redis OSS (cluster mode disabled) (Console)

ElastiCache supports replication when you use the Valkey or Redis OSS engine. To monitor the latency between when data is written to a Valkey or Redis OSS read/write primary cluster and when it is propagated to a read-only secondary cluster, ElastiCache adds to the cluster a special key, `ElastiCacheMasterReplicationTimestamp`. This key is the current Universal Universal Time (UTC) time. Because a Valkey or Redis OSS cluster might be added to a replication group at a later time, this key is included in all Valkey or Redis OSS clusters, even if initially they are not members of a replication group. For more information on replication groups, see High availability using replication groups.

To create a Valkey or Redis OSS (cluster mode disabled) cluster, follow the steps at Creating a Valkey (cluster mode disabled) cluster (Console).

As soon as your cluster's status is *available*, you can grant Amazon EC2 access to it, connect to it, and begin using it. For more information, see Step 3. Authorize access to the cluster and Step 4. Connect to the cluster's node.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console)

With Valkey and Redis OSS, you can create a Valkey or Redis OSS (cluster mode enabled) cluster. Valkey or Redis OSS (cluster mode enabled) clusters support partitioning your data across 1 to 500 shards (API/CLI: node groups) but with some limitations. For a comparison of Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled), see Engine versions and upgrading in ElastiCache.

**To create a Valkey or Redis OSS (cluster mode enabled) cluster using the ElastiCache console**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region that you want to launch this cluster in.
3. Choose **Get started** from the navigation pane.
4. Choose **Create VPC** and follow the steps outlined at Creating a Virtual Private Cloud (VPC).
5. On the ElastiCache dashboard page, choose **Create cluster** and then choose **Create Valkey cluster** or **Create Redis OSS cluster**.
6. Under **Cluster settings**, do the following:
   a. Choose **Configure and create a new cluster**.
   b. For **Cluster mode**, choose **Enabled**.
   c. For **Cluster info** enter a value for **Name**.
   d. (Optional) Enter a value for **Description**.
7. Under **Location**:

   AWS Cloud

   1. For **AWS Cloud**, we recommend you accept the default settings for **Multi-AZ** and **Auto-failover**. For more information, see Minimizing downtime in ElastiCache for Redis OSS with Multi-AZ.
   2. Under **Cluster settings**
      a. For **Engine version**, choose an available version.
      b. For **Port**, use the default port, 6379. If you have a reason to use a different port, enter the port number.
      c. For **Parameter group**, choose a parameter group or create a new one. Parameter groups control the runtime parameters of your cluster. For more information on parameter groups, see Valkey and Redis OSS parameters and Creating an ElastiCache parameter group.

> **Note**
>
> When you select a parameter group to set the engine configuration values, that parameter group is applied to all clusters in the global datastore. On the **Parameter Groups** page, the yes/no **Global** attribute indicates whether a parameter group is part of a global datastore.

d. For **Node type**, choose the down arrow (▼ ).
In the **Change node type** dialog box, choose a value for **Instance family** for the node type that you want. Then choose the node type that you want to use for this cluster, and then choose **Save**.

For more information, see Choosing your node size.

If you choose a data tiering node type, ElastiCache automatically enables data tiering. For more information, see Data tiering in ElastiCache.

e. For **Number of shards**, choose the number of shards (partitions/node groups) that you want for this Valkey or Redis OSS (cluster mode enabled) cluster.

You can change the number of shards in your cluster dynamically. For more information, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters.

f. For **Replicas per shard**, choose the number of read replica nodes that you want in each shard.

The following restrictions exist for Valkey or Redis OSS (cluster mode enabled).

- If you have Multi-AZ enabled, make sure that you have at least one replica per shard.
- The number of replicas is the same for each shard when creating the cluster using the console.
- The number of read replicas per shard is fixed and cannot be changed. If you find you need more or fewer replicas per shard (API/CLI: node group), you must create a new cluster with the new number of replicas. For more information, see Tutorial: Seeding a new node-based cluster with an externally created backup.

3. Under **Connectivity**

a. For **Network type**, choose the IP version(s) this cluster will support.

b. For **Subnet groups**, choose the subnet that you want to apply to this cluster. ElastiCache uses that subnet group to choose a subnet and IP addresses within that subnet to associate with your nodes. ElastiCache clusters require a dual-stack subnet with both IPv4 and IPv6 addresses assigned to them to operate in dual-stack mode and an IPv6-only subnet to operate as IPv6-only.

When creating a new subnet group, enter the **VPC ID** to which it belongs.

Select a **Discovery IP type**. Only the IP adresses of your chosen protocol are returned.

For more information, see:

- Choosing a network type in ElastiCache.
- Create a subnet in your VPC.

If you are Using local zones with ElastiCache, you must create or choose a subnet that is in the local zone.

For more information, see Subnets and subnet groups.

4. For **Availability zone placements**, you have two options:

- **No preference** – ElastiCache chooses the Availability Zone.
- **Specify availability zones** – You specify the Availability Zone for each cluster.

If you chose to specify the Availability Zones, for each cluster in each shard, choose the Availability Zone from the list.

For more information, see Choosing regions and availability zones for ElastiCache.

5. Choose **Next**

6. Under **Advanced Valkey settings** or **Advanced Redis OSS settings** or

- For **Security**:

i. To encrypt your data, you have the following options:

- **Encryption at rest** – Enables encryption of data stored on disk. For more information, see Encryption at Rest.

  > **Note**
  >
  > You have the option to supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key. For more information, see Using customer managed keys from AWS KMS.

- **Encryption in-transit** – Enables encryption of data on the wire. For more information, see encryption in transit. For Valkey 7.2 and above or Redis OSS 6.0 and above, if you enable Encryption in-transit you will be prompted to specify one of the following **Access Control** options:
  - **No Access Control** – This is the default setting. This indicates no restrictions on user access to the cluster.
  - **User Group Access Control List** – Select a user group with a defined set of users that can access the cluster. For more information, see Managing User Groups with the Console and CLI.
  - **AUTH Default User** – An authentication mechanism for a Valkey or Redis OSS server. For more information, see AUTH.
- **AUTH** – An authentication mechanism for Valkey or Redis OSS server. For more information, see AUTH.

> **Note**
>
> RBAC is available on Valkey, and on Redis OSS version 6.0 and later. On earlier Redis OSS versions, AUTH is the sole option.

ii. For **Security groups**, choose the security groups that you want for this cluster. A *security group* acts as a firewall to control network access to your cluster. You can use the default security group for your VPC or create a new one.

For more information on security groups, see Security groups for your VPC in the *Amazon VPC User Guide.*

7. For regularly scheduled automatic backups, select **Enable automatic backups** and then enter the number of days that you want each automatic backup retained before it is automatically deleted. If you don't want regularly scheduled automatic backups, clear the **Enable automatic backups** check box. In either case, you always have the option to create manual backups.

   For more information on backup and restore, see Snapshot and restore.

8. (Optional) Specify a maintenance window. The *maintenance window* is the time, generally an hour in length, each week when ElastiCache schedules system maintenance for your cluster. You can allow ElastiCache to choose the day and time for your maintenance window (*No preference*), or you can choose the day, time, and duration yourself (*Specify maintenance window*). If you choose *Specify maintenance window* from the lists, choose the *Start day*, *Start time*, and *Duration* (in hours) for your maintenance window. All times are UCT times.

   For more information, see Managing ElastiCache cluster maintenance.

9. (Optional) For **Logs**:

   - Under **Log format**, choose either **Text** or **JSON**.
   - Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.
   - Under **Log destination**, choose either **Create new** and enter either your CloudWatch Logs log group name or your Firehose stream name, or choose **Select existing** and then choose either your CloudWatch Logs log group name or your Firehose stream name,

10. For **Tags**, to help you manage your clusters and other ElastiCache resources, you can assign your own metadata to each resource in the form of tags. For mor information, see Tagging your ElastiCache resources.

11. Choose **Next**.

12. Review all your entries and choices, then make any needed corrections. When you're ready, choose **Create**.

On premises

1. For **On premises**, we recommend you leave **Auto-failover** enabled. For more information, see Minimizing downtime in ElastiCache for Redis OSS with Multi-AZ

2. Follow the steps at Using Outposts.

To create the equivalent using the ElastiCache API or AWS CLI instead of the ElastiCache console, see the following:

- API: CreateReplicationGroup
- CLI: create-replication-group

As soon as your cluster's status is *available*, you can grant EC2 access to it, connect to it, and begin using it. For more information, see Step 3. Authorize access to the cluster and Step 4. Connect to the cluster's node.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Creating a cluster (AWS CLI)

To create a cluster using the AWS CLI, use the `create-cache-cluster` command.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Creating a Valkey or Redis OSS (cluster mode disabled) cluster (CLI)

#### Example– A Valkey or Redis OSS (cluster mode disabled) Cluster with no read replicas

The following CLI code creates a Valkey or Redis OSS (cluster mode disabled) cluster with no replicas.

> **Note**
>
> When creating cluster using a data tiering node type, you must pass the `data-tiering-enabled` parameter.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
--cache-cluster-id my-cluster \
--cache-node-type cache.r4.large \
--engine redis \
--num-cache-nodes 1 \
--cache-parameter-group default.redis6.x \
--snapshot-arns arn:aws:s3:::amzn-s3-demo-bucket/snapshot.rdb
```

For Windows:

```
aws elasticache create-cache-cluster ^
--cache-cluster-id my-cluster ^
--cache-node-type cache.r4.large ^
--engine redis ^
--num-cache-nodes 1 ^
```

```
--cache-parameter-group default.redis6.x ^
--snapshot-arns arn:aws:s3:::amzn-s3-demo-bucket/snapshot.rdb
```

### Creating a Valkey or Redis OSS (cluster mode enabled) cluster (AWS CLI)

Valkey or Redis OSS (cluster mode enabled) clusters (API/CLI: replication groups) cannot be created using the `create-cache-cluster` operation. To create a Valkey or Redis OSS (cluster mode enabled) cluster (API/CLI: replication group), see Creating a Valkey or Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI).

For more information, see the AWS CLI for ElastiCache reference topic `create-replication-group`.

### Creating a cluster for Valkey or Redis OSS (ElastiCache API)

To create a cluster using the ElastiCache API, use the `CreateCacheCluster` action.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Topics

- Creating a Valkey or Redis OSS (cluster mode disabled) cluster (ElastiCache API)
- Creating a cluster in Valkey or Redis OSS (cluster mode enabled) (ElastiCache API)

### Creating a Valkey or Redis OSS (cluster mode disabled) cluster (ElastiCache API)

The following code creates a Valkey or Redis OSS (cluster mode disabled) cluster (ElastiCache API).

Line breaks are added for ease of reading.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=CreateCacheCluster
    &CacheClusterId=my-cluster
    &CacheNodeType=cache.r4.large
    &CacheParameterGroup=default.redis7
    &Engine=redis
```

```
    &EngineVersion=7.1
    &NumCacheNodes=1
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &SnapshotArns.member.1=arn%3Aaws%3As3%3A%3A%3AmyS3Bucket%2Fdump.rdb
    &Timestamp=20150508T220302Z
    &Version=2015-02-02
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Credential=<credential>
    &X-Amz-Date=20150508T220302Z
    &X-Amz-Expires=20150508T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Signature=<signature>
```

**Creating a cluster in Valkey or Redis OSS (cluster mode enabled) (ElastiCache API)**

Valkey or Redis OSS (cluster mode enabled) clusters (API/CLI: replication groups) cannot be created using the `CreateCacheCluster` operation. To create a Valkey or Redis OSS (cluster mode enabled) cluster (API/CLI: replication group), see Creating a replication group in Valkey or Redis OSS (Cluster Mode Enabled) from scratch (ElastiCache API).

For more information, see the ElastiCache API reference topic `CreateReplicationGroup`.

## Creating a cluster for Memcached

The following examples show how to create a cluster using the AWS Management Console, AWS CLI and ElastiCache API.

**Creating a Memcached cluster (console)**

When you use the Memcached engine, Amazon ElastiCache supports horizontally partitioning your data over multiple nodes. Memcached enables auto discovery so you don't need to keep track of the endpoints for each node. Memcached tracks each node's endpoint, updating the endpoint list as nodes are added and removed. All your application needs to interact with the cluster is the configuration endpoint.

To create a Memcached cluster via the console, follow the steps at Creating a Valkey (cluster mode disabled) cluster (Console). When you reach step five, select **Create Memcached cache**.

As soon as your cluster's status is *available*, you can grant Amazon EC2 access to it, connect to it, and begin using it. For more information, see the similar steps Step 3. Authorize access to the cluster and Step 4. Connect to the cluster's node.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Creating a cluster (AWS CLI)

To create a cluster using the AWS CLI, use the `create-cache-cluster` command.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not actively using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

### Creating a Memcached Cache Cluster (AWS CLI)

The following CLI code creates a Memcached cluster with 3 nodes.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
--cache-cluster-id my-cluster \
--cache-node-type cache.r4.large \
--engine memcached \
--engine-version 1.4.24 \
--cache-parameter-group default.memcached1.4 \
--num-cache-nodes 3
```

For Windows:

```
aws elasticache create-cache-cluster ^
--cache-cluster-id my-cluster ^
--cache-node-type cache.r4.large ^
--engine memcached ^
--engine-version 1.4.24 ^
--cache-parameter-group default.memcached1.4 ^
```

```
--num-cache-nodes 3
```

### Creating a cluster for Memcached (ElastiCache API)

To create a cluster using the ElastiCache API, use the `CreateCacheCluster` action.

> **Important**
>
> As soon as your cluster becomes available, you're billed for each hour or partial hour that the cluster is active, even if you're not using it. To stop incurring charges for this cluster, you must delete it. See Deleting a cluster in ElastiCache.

#### Topics

- Creating a Memcached cluster (ElastiCache API)

### Creating a Memcached cluster (ElastiCache API)

The following code creates a Memcached cluster with 3 nodes (ElastiCache API).

Line breaks are added for ease of reading.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=CreateCacheCluster
    &CacheClusterId=my-cluster
    &CacheNodeType=cache.r4.large
    &Engine=memcached
    &NumCacheNodes=3
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150508T220302Z
    &Version=2015-02-02
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Credential=<credential>
    &X-Amz-Date=20150508T220302Z
    &X-Amz-Expires=20150508T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Signature=<signature>
```

# Viewing an ElastiCache cluster's details

You can view detail information about one or more clusters using the ElastiCache console, AWS CLI, or ElastiCache API.

### Viewing a Memcached cluster's details (Console)

You can view the details of a Memcached cluster using the ElastiCache console, the AWS CLI for ElastiCache, or the ElastiCache API.

The following procedure details how to view the details of a Memcached cluster using the ElastiCache console.

### To view a Memcached cluster's details

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region you are interested in.
3. In the ElastiCache engine dashboard, choose **Memcached**. Doing this displays a list of all your clusters that are running on the Memcached engine.
4. To see details of a cluster, choose the box to the left of the cluster's name.
5. To view node information, choose the **Nodes** tab, which displays information on node(s) status and endpoint.
6. To view metrics, choose the **Metrics** tab, which displays the relevant metrics for all nodes in the cluster. For more information, see Monitoring use with CloudWatch Metrics
7. Choose the **Network and security** tab to view details on the cluster's network connectivity and subnet group configuration and the VPC security group. For more information, see Subnets and subnet groups.
8. Choose the **Maintenance** tab to view details on the cluster's maintenance settings. For more information, see Managing ElastiCache cluster maintenance.
9. Choose the **Tags** tab to view details on any tags applied to cluster resources. For more information, see Tagging your ElastiCache resources.

### Viewing Valkey or Redis OSS (Cluster Mode Disabled) details (Console)

You can view the details of a Valkey or Redis OSS (cluster mode disabled) cluster using the ElastiCache console, the AWS CLI for ElastiCache, or the ElastiCache API.

The following procedure details how to view the details of a Valkey or Redis OSS (cluster mode disabled) cluster using the ElastiCache console.

**To view a Valkey or Redis OSS (cluster mode disabled) cluster's details**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the ElastiCache engine dashboard, choose either **Valkey**or **Redis OSS** to display a list of all your clusters that are running on that engine.
3. To see details of a cluster, select the check box to the left of the cluster's name. Make sure that you select a cluster running the Valkey or Redis OSS engine, not Clustered Valkey or Clustered Redis OSS. Doing this displays details about the cluster, including the cluster's primary endpoint.
4. To view node information:
    a. Choose the cluster's name.
    b. Choose the **Shards and nodes** tab. Doing this displays details about each node, including the node's endpoint which you need to use to read from the cluster.
5. To view metrics, choose the **Metrics** tab, which displays the relevant metrics for all nodes in the cluster. For more information, see Monitoring use with CloudWatch Metrics
6. To view logs, choose the **Logs** tab, which indicates if the cluster is using Slow logs or Engine logs and provides relevant details. For more information, see Log delivery.
7. Choose the **Network and security** tab to view details on the cluster's network connectivity and subnet group configuration. For more information, see Subnets and subnet groups.
8. Choose the **Maintenance** tab to view details on the cluster's maintenance settings. For more information, see Managing ElastiCache cluster maintenance.
9. Choose the **Service updates** tab to view details on any available service updates along with their recommended apply-by date. For more information, see Service updates in ElastiCache.
10. Choose the **Tags** tab to view details on any tags applied to cluster resources. For more information, see Tagging your ElastiCache resources.

### Viewing details for a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console)

You can view the details of a Valkey or Redis OSS (cluster mode enabled) cluster using the ElastiCache console, the AWS CLI for ElastiCache, or the ElastiCache API.

The following procedure details how to view the details of a Valkey or Redis OSS (cluster mode enabled) cluster using the ElastiCache console.

**To view a Valkey or Redis OSS (cluster mode enabled) cluster's details**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region you are interested in.
3. In the ElastiCache engine dashboard, choose **Valkey** or **Redis OSS** to display a list of all your clusters that are running on that engine.
4. To see details of a Valkey or Redis OSS (cluster mode enabled) cluster, choose the box to the left of the cluster's name. Make sure you choose a cluster running the Valkey or Clustered Redis OSS engine.

   The screen expands below the cluster and display details about the cluster, including the cluster's configuration endpoint.
5. To see a listing of the cluster's shards and the number of nodes in each shard, choose the **Shards and nodes** tab.
6. To view specific information on a node:

   - Choose the shard's ID.

     Doing this displays information about each node, including each node's endpoint that you need to use to read data from the cluster.
7. To view metrics, choose the **Metrics** tab, which displays the relevant metrics for all nodes in the cluster. For more information, see Monitoring use with CloudWatch Metrics
8. To view logs, choose the **Logs** tab, which indicates if the cluster is using Slow logs or Engine logs and provides relevant details. For more information, see Log delivery.
9. Choose the **Network and security** tab to view details on the cluster's network connectivity and subnet group configuration, the VPC security group and what, if any, encryption method is enabled on the cluster. For more information, see Subnets and subnet groups and Data security in Amazon ElastiCache.
10. Choose the **Maintenance** tab to view details on the cluster's maintenance settings. For more information, see Managing ElastiCache cluster maintenance.
11. Choose the **Service updates** tab to view details on any available service updates along with their recommended apply-by date. For more information, see Service updates in ElastiCache.

12. Choose the **Tags** tab to view details on any tags applied to cluster resources. For more information, see Tagging your ElastiCache resources.

### Viewing an ElastiCache cluster's details (AWS CLI)

The following code lists the details for *my-cluster*:

```
aws elasticache describe-cache-clusters --cache-cluster-id my-cluster
```

Replace *my-cluster* with the name of your cluster in a case where the cluster is created with 1 cache node and 0 shards using the `create-cache-cluster` command.

```
{
    "CacheClusters": [
        {
            "CacheClusterStatus": "available",
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "wed:12:00-wed:13:00",
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "08:30-09:30",
            "TransitEncryptionEnabled": false,
            "AtRestEncryptionEnabled": false,
            "CacheClusterId": "my-cluster1",
            "CacheClusterCreateTime": "2018-02-26T21:06:43.420Z",
            "PreferredAvailabilityZone": "us-west-2c",
            "AuthTokenEnabled": false,
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
           "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis7"
            },
```

```
            "SnapshotRetentionLimit": 0,
            "AutoMinorVersionUpgrade": true,
            "EngineVersion": "7.1",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        }
```

```
{
    "CacheClusters": [
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": false,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:13:24.250Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": false,
            "PreferredAvailabilityZone": "us-west-2a",
            "TransitEncryptionEnabled": false,
            "ReplicationGroupId": "my-cluster2",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "sun:08:30-sun:09:30",
            "CacheClusterId": "my-cluster2-001",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
            "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis6.x"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "6.0",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
```

```
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": false,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:13:24.250Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": false,
            "PreferredAvailabilityZone": "us-west-2b",
            "TransitEncryptionEnabled": false,
            "ReplicationGroupId": "my-cluster2",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "sun:08:30-sun:09:30",
            "CacheClusterId": "my-cluster2-002",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
            "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis6.x"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "6.0",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
```

```
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": false,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:13:24.250Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": false,
            "PreferredAvailabilityZone": "us-west-2c",
            "TransitEncryptionEnabled": false,
            "ReplicationGroupId": "my-cluster2",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "sun:08:30-sun:09:30",
            "CacheClusterId": "my-cluster2-003",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
            "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis7"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "7.1",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        }
```

```
{
    "CacheClusters": [
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": true,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
```

```
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:17:01.439Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": true,
            "PreferredAvailabilityZone": "us-west-2a",
            "TransitEncryptionEnabled": true,
            "ReplicationGroupId": "my-cluster3",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "thu:11:00-thu:12:00",
            "CacheClusterId": "my-cluster3-0001-001",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
            "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis6.x.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "6.0",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": true,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:17:01.439Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": true,
            "PreferredAvailabilityZone": "us-west-2b",
            "TransitEncryptionEnabled": true,
            "ReplicationGroupId": "my-cluster3",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "thu:11:00-thu:12:00",
```

```
            "CacheClusterId": "my-cluster3-0001-002",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
             "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis7.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "7.1",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": true,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:17:01.439Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": true,
            "PreferredAvailabilityZone": "us-west-2c",
            "TransitEncryptionEnabled": true,
            "ReplicationGroupId": "my-cluster3",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "thu:11:00-thu:12:00",
            "CacheClusterId": "my-cluster3-0001-003",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
             "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis6.x.cluster.on"
            },
```

```
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "6.0",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": true,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:17:01.439Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": true,
            "PreferredAvailabilityZone": "us-west-2b",
            "TransitEncryptionEnabled": true,
            "ReplicationGroupId": "my-cluster3",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "thu:11:00-thu:12:00",
            "CacheClusterId": "my-cluster3-0002-001",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
             "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis6.x.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "6.0",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
```

```
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": true,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterCreateTime": "2018-02-26T21:17:01.439Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": true,
            "PreferredAvailabilityZone": "us-west-2c",
            "TransitEncryptionEnabled": true,
            "ReplicationGroupId": "my-cluster3",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "thu:11:00-thu:12:00",
            "CacheClusterId": "my-cluster3-0002-002",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
             "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis7.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "7.1",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        },
        {
            "SecurityGroups": [
                {
                    "Status": "active",
                    "SecurityGroupId": "sg-dbe93fa2"
                }
            ],
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "AuthTokenEnabled": true,
            "CacheSubnetGroupName": "default",
            "SnapshotWindow": "12:30-13:30",
            "AutoMinorVersionUpgrade": true,
```

```
            "CacheClusterCreateTime": "2018-02-26T21:17:01.439Z",
            "CacheClusterStatus": "available",
            "AtRestEncryptionEnabled": true,
            "PreferredAvailabilityZone": "us-west-2a",
            "TransitEncryptionEnabled": true,
            "ReplicationGroupId": "my-cluster3",
            "Engine": "redis",
            "PreferredMaintenanceWindow": "thu:11:00-thu:12:00",
            "CacheClusterId": "my-cluster3-0002-003",
            "PendingModifiedValues": {},
            "CacheNodeType": "cache.r4.large",
             "DataTiering": "disabled",
            "CacheParameterGroup": {
                "CacheNodeIdsToReboot": [],
                "ParameterApplyStatus": "in-sync",
                "CacheParameterGroupName": "default.redis6.x.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "6.0",
            "CacheSecurityGroups": [],
            "NumCacheNodes": 1
        }
    ]
}
```

In a case where the cluster is created using the AWS Management Console (cluster node enabled or disabled with 1 or more shards), use the following command to describe the cluster's details (replace *my-cluster* with the name of the replication group (name of your cluster):

```
aws elasticache describe-replication-groups --replication-group-id my-cluster
```

For more information, see the AWS CLI for ElastiCache topic describe-cache-clusters.

**Viewing an ElastiCache cluster's details (ElastiCache API)**

You can view the details for a cluster using the ElastiCache API `DescribeCacheClusters` action. If the `CacheClusterId` parameter is included, details for the specified cluster are returned. If the `CacheClusterId` parameter is omitted, details for up to `MaxRecords` (default 100) clusters are returned. The value for `MaxRecords` cannot be less than 20 or greater than 100.

The following code lists the details for `my-cluster`.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheClusters
   &CacheClusterId=my-cluster
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

The following code list the details for up to 25 clusters.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheClusters
   &MaxRecords=25
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

For more information, see the ElastiCache API reference topic `DescribeCacheClusters`.

# Modifying an ElastiCache cluster

In addition to adding or removing nodes from an ElastiCache cluster, there can be times where you need to make other changes such as adding a security group, changing the maintenance window or a parameter group.

We recommend that you have your maintenance window fall at the time of lowest usage. Thus it might need modification from time to time.

When you change a cluster's parameters, the change is applied to the cluster either immediately or after the cluster is restarted. This is true whether you change the cluster's parameter group itself or a parameter value within the cluster's parameter group. To determine when a particular parameter change is applied, see the **Changes Take Effect** section of the **Details** column in the tables for Memcached specific parameters and Valkey and Redis OSS parameters. For information on rebooting a cluster's nodes, see Rebooting nodes.

**Using the ElastiCache AWS Management Console**

**To modify a cluster**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region where the cluster that you want to modify is located.
3. In the navigation pane, choose the engine running on the cluster that you want to modify.

   A list of the chosen engine's clusters appears.
4. In the list of clusters, for the cluster that you want to modify, choose its name.
5. Choose **Actions** and then choose **Modify**.

   The **Modify Cluster** window appears.
6. In the **Modify Cluster** window, make the modifications that you want. Options include:

   - Description
   - Cluster mode - To modify cluster mode from **Disabled** to **Enabled**, you must first set the cluster mode to **Compatible**.

     Compatible mode allows your Valkey or Redis OSS clients to connect using both cluster mode enabled and cluster mode disabled. After you migrate all Valkey or Redis OSS clients

to use cluster mode enabled, you can then complete cluster mode configuration and set the cluster mode to **Enabled**.

- Engine Version Compatibility

> **Important**
>
> You can upgrade to newer engine versions. If you upgrade major engine versions, for example from 5.0.6 to 6.0, you need to select a parameter group family that is compatible with the new engine version. For more information on doing so, see Version Management for ElastiCache. However, you can't downgrade to older engine versions except by deleting the existing cluster and creating it again.

- VPC Security Group(s)
- Parameter Group
- Node Type

> **Note**
>
> If the cluster is using a data tiering node type, you can only choose a different node size from within the same data tiering node family. If you choose a data tiering node type, data tiering is automatically enabled. For more information, see Data tiering.

- Multi-AZ
- Auto failover (cluster mode disabled only)
- Enable Automatic Backups
- Backup Node Id
- Backup Retention Period
- Backup Window
- Topic for SNS Notification

- Memcached Engine Version Compatibility
- Network type

> **Note**
>
> If you are switching from IPv4 to IPv6, you must select or create subnet groups compatible with IPv6. For more information, see Choosing a network type in ElastiCache.

- VPC Security Group(s)
- Parameter Group
- Maintenance Window
- Topic for SNS Notification

The **Apply Immediately** box applies only to engine version and node type modifications. To apply changes immediately, choose the **Apply Immediately** check box. If this box is not chosen, engine version modifications are applied during the next maintenance window. Other modifications, such as changing the maintenance window, are applied immediately.

**To enable/disable log delivery for Redis**

1. From the list of clusters, choose the cluster you want to modify. Choose the **Cluster name** and not the checkbox beside it.
2. On the **Cluster details** page, choose the **Logs** tab.
3. To enable or disable slow logs, choose either **Enable** or **Disable**.

   If you choose enable:

   a. Under **Log format**, choose either **JSON** or **Text**.

   b. Under **Log destination type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.

   c. Under **Log destination**, you can choose **Create new** and enter either your CloudWatchLogs log group name or your Kinesis Data Firehose stream name. You can also choose **Select existing** and then choose either your CloudWatchLogs log group name or your Kinesis Data Firehose stream name.

   d. Choose **Enable**.

**To change your configuration for Redis:**

1. Choose **Modify**.
2. Under **Log format**, choose either **JSON** or **Text**.
3. Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.
4. Under **Log destination**, choose either **Create new** and enter your CloudWatchLogs log group name or your Kinesis Data Firehose stream name. Or choose **Select existing** and then choose your CloudWatchLogs log group name or your Kinesis Data Firehose stream name.

### Using the AWS CLI with ElastiCache

You can modify an existing cluster using the AWS CLI `modify-cache-cluster` operation. To modify a cluster's configuration value, specify the cluster's ID, the parameter to change and the parameter's new value. The following example changes the maintenance window for a cluster named `my-cluster` and applies the change immediately.

> **Important**
>
> You can upgrade to newer Memcached engine versions. For more information on doing so, see Version Management for ElastiCache. However, you can't downgrade to older engine versions except by deleting the existing cluster and creating it again.

> **Important**
>
> You can upgrade to newer Valkey or Redis OSS engine versions. If you upgrade major engine versions, for example from Redis OSS 5.0.6 to Redis OSS 6.0, you need to select a parameter group family that is compatible with the new engine version. For more information on doing so, see Version Management for ElastiCache. However, you can't downgrade to older engine versions except by deleting the existing cluster and creating it again.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-cluster \
    --cache-cluster-id my-cluster \
```

```
    --preferred-maintenance-window sun:23:00-mon:02:00
```

For Windows:

```
aws elasticache modify-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --preferred-maintenance-window sun:23:00-mon:02:00
```

The `--apply-immediately` parameter applies only to modifications in node type, engine version, and changing the number of nodes in a cluster. If you want to apply any of these changes immediately, use the `--apply-immediately` parameter. If you prefer postponing these changes to your next maintenance window, use the `--no-apply-immediately` parameter. Other modifications, such as changing the maintenance window, are applied immediately.

For more information, see the AWS CLI for ElastiCache topic `modify-cache-cluster`.

**Using the ElastiCache API**

You can modify an existing cluster using the ElastiCache API `ModifyCacheCluster` operation. To modify a cluster's configuration value, specify the cluster's ID, the parameter to change and the parameter's new value. The following example changes the maintenance window for a cluster named `my-cluster` and applies the change immediately.

**Important**

You can upgrade to newer Memcached engine versions. For more information on doing so, see Version Management for ElastiCache. However, you can't downgrade to older engine versions except by deleting the existing cluster and creating it again.

**Important**

You can upgrade to newer Valkey or Redis OSS engine versions. If you upgrade major engine versions, for example from Redis OSS 5.0.6 to Redis OSS 6.0, you need to select a parameter group family that is compatible with the new engine version. For more information on doing so, see Version Management for ElastiCache. However, you can't downgrade to older engine versions except by deleting the existing cluster and creating it again.

Line breaks are added for ease of reading.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyCacheCluster
    &CacheClusterId=my-cluster
    &PreferredMaintenanceWindow=sun:23:00-mon:02:00
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150901T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20150202T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20150901T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

The `ApplyImmediately` parameter applies only to modifications in node type, engine version, and changing the number of nodes in a cluster. If you want to apply any of these changes immediately, set the `ApplyImmediately` parameter to `true`. If you prefer postponing these changes to your next maintenance window, set the `ApplyImmediately` parameter to `false`. Other modifications, such as changing the maintenance window, are applied immediately.

For more information, see the ElastiCache API reference topic `ModifyCacheCluster`.

# Adding nodes to an ElastiCache cluster

Adding nodes to a Memcached cluster increases the number of your cluster's partitions. When you change the number of partitions in a cluster some of your key spaces need to be remapped so that they are mapped to the right node. Remapping key spaces temporarily increases the number of cache misses on the cluster. For more information, see Configuring your ElastiCache client for efficient load balancing (Memcached).

To reconfigure your Valkey or Redis OSS (cluster mode enabled) cluster, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters

You can use the ElastiCache Management Console, the AWS CLI or ElastiCache API to add nodes to your cluster.

### Using the ElastiCache AWS Management Console

If you want to add a node to a single-node Valkey or Redis OSS (cluster mode disabled) cluster (one without replication enabled), it's a two-step process: first add replication, and then add a replica node.

### Topics

- To add replication to a Valkey or Redis OSS cluster with no shards
- To add nodes to an ElastiCache cluster (console)

The following procedure adds replication to a single-node Valkey or Redis OSS that does not have replication enabled. When you add replication, the existing node becomes the primary node in the replication-enabled cluster. After replication is added, you can add up to 5 replica nodes to the cluster.

### To add replication to a Valkey or Redis OSS cluster with no shards

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.

   A list of clusters running that engine is displayed.
3. Choose the name of a cluster, not the box to the left of the cluster's name, that you want to add nodes to.

The following is true of a Redis OSS cluster that does not have replication enabled:

- It is running Redis OSS, not Clustered Redis OSS.
- It has zero shards.

  If the cluster has any shards, replication is already enabled on it and you can continue at To add nodes to an ElastiCache cluster (console).

4. Choose **Add replication**.
5. In **Add Replication**, enter a description for this replication-enabled cluster.
6. Choose **Add**.

   As soon as the cluster's status returns to *available* you can continue at the next procedure and add replicas to the cluster.

**To add nodes to an ElastiCache cluster (console)**

The following procedure can be used to add nodes to a cluster.

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose the engine running on the cluster that you want to add nodes to.

   A list of clusters running the chosen engine appears.
3. From the list of clusters, for the cluster that you want to add a node to, choose its name.

   If your cluster is a Valkey or Redis OSS (cluster mode enabled) cluster, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters.

   If your cluster is a Valkey or Redis OSS (cluster mode disabled) cluster with zero shards, first complete the steps at To add replication to a Valkey or Redis OSS cluster with no shards.
4. Choose **Add node**.
5. Complete the information requested in the **Add Node** dialog box.
6. Choose the **Apply Immediately - Yes** button to add this node immediately, or choose **No** to add this node during the cluster's next maintenance window.

**Impact of New Add and Remove Requests on Pending Requests**

| Scenarios | Pending Operation | New Request | Results |
|---|---|---|---|
| Scenario 1 | Delete | Delete | The new delete request, pending or immediate, replaces the pending delete request.<br><br>For example, if nodes 0001, 0003, and 0007 are pending deletion and a new request to delete nodes 0002 and 0004 is issued, only nodes 0002 and 0004 will be deleted. Nodes 0001, 0003, and 0007 will not be deleted. |
| Scenario 2 | Delete | Create | The new create request, pending or immediate, replaces the pending delete request.<br><br>For example, if nodes 0001, 0003, and 0007 are pending deletion and a new request to create a node is issued, a new node will be created and nodes 0001, 0003, and 0007 will not be deleted. |
| Scenario 3 | Create | Delete | The new delete request, pending or immediate, replaces the pending create request.<br><br>For example, if there is a pending request to create two nodes and a new request is issued to delete node 0003, no new nodes will be created and node 0003 will be deleted. |

| Scenarios | Pending Operation | New Request | Results |
|---|---|---|---|
| Scenario 4 | Create | Create | The new create request is added to the pending create request.<br><br>For example, if there is a pending request to create two nodes and a new request is issued to create three nodes, the new requests is added to the pending request and five nodes will be created.<br><br>**Important**<br>If the new create request is set to **Apply Immediately - Yes**, all create requests are performed immediately. If the new create request is set to **Apply Immediately - No**, all create requests are pending. |

To determine what operations are pending, choose the **Description** tab and check to see how many pending creations or deletions are shown. You cannot have both pending creations and pending deletions.

7. Choose the **Add** button.

   After a few moments, the new nodes should appear in the nodes list with a status of **creating**. If they don't appear, refresh your browser page. When the node's status changes to *available* the new node is able to be used.

### Using the AWS CLI with ElastiCache

To add nodes to a cluster using the AWS CLI, use the AWS CLI operation `modify-cache-cluster` with the following parameters:

- `--cache-cluster-id` The ID of the cluster that you want to add nodes to.
- `--num-cache-nodes` The `--num-cache-nodes` parameter specifies the number of nodes that you want in this cluster after the modification is applied. To add nodes to this cluster, `--num-`

`cache-nodes` must be greater than the current number of nodes in this cluster. If this value is less than the current number of nodes, ElastiCache expects the parameter `cache-node-ids-to-remove` and a list of nodes to remove from the cluster. For more information, see Using the AWS CLI with ElastiCache.

- `--apply-immediately` or `--no-apply-immediately` which specifies whether to add these nodes immediately or at the next maintenance window.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-cluster \
    --cache-cluster-id my-cluster \
    --num-cache-nodes 5 \
    --apply-immediately
```

For Windows:

```
aws elasticache modify-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --num-cache-nodes 5 ^
    --apply-immediately
```

This operation produces output similar to the following (JSON format):

```
{
    "CacheCluster": {
        "Engine": "memcached",
        "CacheParameterGroup": {
            "CacheNodeIdsToReboot": [],
            "CacheParameterGroupName": "default.memcached1.4",
            "ParameterApplyStatus": "in-sync"
        },
        "CacheClusterId": "my-cluster",
        "PreferredAvailabilityZone": "us-west-2b",
        "ConfigurationEndpoint": {
            "Port": 11211,
            "Address": "rlh-mem000.7alc7bf-example.cfg.usw2.cache.amazonaws.com"
        },
        "CacheSecurityGroups": [],
        "CacheClusterCreateTime": "2016-09-21T16:28:28.973Z",
        "AutoMinorVersionUpgrade": true,
        "CacheClusterStatus": "modifying",
```

```
        "NumCacheNodes": 2,
        "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
        "SecurityGroups": [
            {
                "Status": "active",
                "SecurityGroupId": "sg-dbe93fa2"
            }
        ],
        "CacheSubnetGroupName": "default",
        "EngineVersion": "1.4.24",
        "PendingModifiedValues": {
            "NumCacheNodes": 5
        },
        "PreferredMaintenanceWindow": "sat:09:00-sat:10:00",
        "CacheNodeType": "cache.m5.large",
         "DataTiering": "disabled",
    }
}
```

For more information, see the AWS CLI topic `modify-cache-cluster`.

**Using the AWS CLI with ElastiCache**

If you want to add nodes to an existing Valkey or Redis OSS (cluster mode disabled) cluster that does not have replication enabled, you must first create the replication group specifying the existing cluster as the primary. For more information, see Creating a replication group using an available Valkey or Redis OSS cluster (AWS CLI). After the replication group is *available*, you can continue with the following process.

To add nodes to a cluster using the AWS CLI, use the AWS CLI operation `increase-replica-count` with the following parameters:

- `--replication-group-id` The ID of the replicationn group that you want to add nodes to.
- `--new-replica-count` specifies the number of nodes that you want in this replication group after the modification is applied. To add nodes to this cluster, `--new-replica-count` must be greater than the current number of nodes in this cluster.
- `--apply-immediately` or `--no-apply-immediately` which specifies whether to add these nodes immediately or at the next maintenance window.

For Linux, macOS, or Unix:

```
aws elasticache increase-replica-count \
    --replication-group-id my-replication-group \
    --new-replica-count 4 \
    --apply-immediately
```

For Windows:

```
aws elasticache increase-replica-count ^
    --replication-group-id my-replication-group ^
    --new-replica-count 4 ^
    --apply-immediately
```

This operation produces output similar to the following (JSON format):

```
{
    "ReplicationGroup": {
        "ReplicationGroupId": "node-test",
        "Description": "node-test",
        "Status": "modifying",
        "PendingModifiedValues": {},
        "MemberClusters": [
            "node-test-001",
            "node-test-002",
            "node-test-003",
            "node-test-004",
            "node-test-005"
        ],
        "NodeGroups": [
            {
                "NodeGroupId": "0001",
                "Status": "modifying",
                "PrimaryEndpoint": {
                    "Address": "node-test.zzzzzz.ng.0001.usw2.cache.amazonaws.com",
                    "Port": 6379
                },
                "ReaderEndpoint": {
                    "Address": "node-test.zzzzzz.ng.0001.usw2.cache.amazonaws.com",
                    "Port": 6379
                },
                "NodeGroupMembers": [
                    {
                        "CacheClusterId": "node-test-001",
```

```
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-001.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2a",
                        "CurrentRole": "primary"
                    },
                    {
                        "CacheClusterId": "node-test-002",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-002.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2c",
                        "CurrentRole": "replica"
                    },
                    {
                        "CacheClusterId": "node-test-003",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-003.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CurrentRole": "replica"
                    }
                ]
            }
        ],
        "SnapshottingClusterId": "node-test-002",
        "AutomaticFailover": "enabled",
        "MultiAZ": "enabled",
        "SnapshotRetentionLimit": 1,
        "SnapshotWindow": "07:30-08:30",
        "ClusterEnabled": false,
        "CacheNodeType": "cache.r5.large",
         "DataTiering": "disabled",
        "TransitEncryptionEnabled": false,
        "AtRestEncryptionEnabled": false,
```

```
        "ARN": "arn:aws:elasticache:us-west-2:123456789012:replicationgroup:node-test"
    }
}
```

For more information, see the AWS CLI topic `increase-replica-count`.

**Using the ElastiCache API**

If you want to add nodes to an existing Valkey or Redis OSS (cluster mode disabled) cluster that does not have replication enabled, you must first create the replication group specifying the existing cluster as the Primary. For more information, see Adding replicas to a standalone Valkey or Redis OSS (Cluster Mode Disabled) cluster (ElastiCache API). After the replication group is *available*, you can continue with the following process.

**To add nodes to a cluster (ElastiCache API)**

- Call the `IncreaseReplicaCount` API operation with the following parameters:
  - `ReplicationGroupId` The ID of the cluster that you want to add nodes to.
  - `NewReplicaCount` The `NewReplicaCount` parameter specifies the number of nodes that you want in this cluster after the modification is applied. To add nodes to this cluster, `NewReplicaCount` must be greater than the current number of nodes in this cluster. If this value is less than the current number of nodes, use the `DecreaseReplicaCount` API with the number of nodes to remove from the cluster.
  - `ApplyImmediately` Specifies whether to add these nodes immediately or at the next maintenance window.
  - `Region` Specifies the AWS Region of the cluster that you want to add nodes to.

  The following example shows a call to add nodes to a cluster.

  **Example**

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=IncreaseReplicaCount
    &ApplyImmediately=true
    &NumCacheNodes=4
    &ReplicationGroupId=my-replication-group
    &Region=us-east-2
    &Version=2014-12-01
    &SignatureVersion=4
```

```
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ElastiCache API topic `IncreaseReplicaCount`.

**Using the ElastiCache API**

**To add nodes to a cluster (ElastiCache API)**

- Call the `ModifyCacheCluster` API operation with the following parameters:
  - `CacheClusterId` The ID of the cluster that you want to add nodes to.
  - `NumCacheNodes` The `NumCachNodes` parameter specifies the number of nodes that you want in this cluster after the modification is applied. To add nodes to this cluster, `NumCacheNodes` must be greater than the current number of nodes in this cluster. If this value is less than the current number of nodes, ElastiCache expects the parameter `CacheNodeIdsToRemove` with a list of nodes to remove from the cluster (see Using the ElastiCache API with Memcached).
  - `ApplyImmediately` Specifies whether to add these nodes immediately or at the next maintenance window.
  - `Region` Specifies the AWS Region of the cluster that you want to add nodes to.

  The following example shows a call to add nodes to a cluster.

  **Example**

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyCacheCluster
    &ApplyImmediately=true
    &NumCacheNodes=5
 &CacheClusterId=my-cluster
 &Region=us-east-2
    &Version=2014-12-01
```

```
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ElastiCache API topic `ModifyCacheCluster`.

# Removing nodes from an ElastiCache cluster

You can delete a node from a Valkey, Memcached, or Redis OSS cluster using the AWS Management Console, the AWS CLI, or the ElastiCache API.

> **Note**
>
> Each time you change the number of nodes in a Memcached cluster, you must re-map at least some of your keyspace so it maps to the correct node. For more detailed information on load balancing a Memcached cluster, see Configuring your ElastiCache client for efficient load balancing (Memcached).

### Using the ElastiCache AWS Management Console

### To remove nodes from a cluster (console)

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region of the cluster that you want to remove nodes from.
3. In the navigation pane, choose the engine running on the cluster that you want to remove a node.

   A list of clusters running the chosen engine appears.
4. From the list of clusters, choose the cluster name from which you want to remove a node.

   A list of the cluster's nodes appears.
5. Choose the box to the left of the node ID for the node that you want to remove. Using the ElastiCache console, you can only delete one node at a time, so choosing multiple nodes means that you can't use the **Delete node** button.

   The *Delete Node* page appears.
6. To delete the node, complete the **Delete Node** page and choose **Delete Node**. To keep the node, choose **Cancel**.

> **Important**
>
> With Valkey or Redis OSS, if you're deleting the node results in the cluster which are no longer Multi-AZ compliant, make sure to first clear the **Multi-AZ** check box and then delete the node. If you clear the **Multi-AZ** check box, you can choose to enable **Auto failover**.

### Impact of New Add and Remove Requests on Pending Requests

| Scenarios | Pending Operation | New Request | Results |
|---|---|---|---|
| Scenario 1 | Delete | Delete | The new delete request, pending or immediate, replaces the pending delete request.<br><br>For example, if nodes 0001, 0003, and 0007 are pending deletion and a new request to delete nodes 0002 and 0004 is issued, only nodes 0002 and 0004 will be deleted. Nodes 0001, 0003, and 0007 will not be deleted. |
| Scenario 2 | Delete | Create | The new create request, pending or immediate, replaces the pending delete request.<br><br>For example, if nodes 0001, 0003, and 0007 are pending deletion and a new request to create a node is issued, a new node will be created and nodes 0001, 0003, and 0007 will not be deleted. |
| Scenario 3 | Create | Delete | The new delete request, pending or immediate, replaces the pending create request.<br><br>For example, if there is a pending request to create two nodes and a new request is issued to delete node 0003, no new nodes will be created and node 0003 will be deleted. |

| Scenarios | Pending Operation | New Request | Results |
| --- | --- | --- | --- |
| Scenario 4 | Create | Create | The new create request is added to the pending create request.<br><br>For example, if there is a pending request to create two nodes and a new request is issued to create three nodes, the new requests is added to the pending request and five nodes will be created.<br><br>**Important**<br>If the new create request is set to **Apply Immediately - Yes**, all create requests are performed immediately. If the new create request is set to **Apply Immediately - No**, all create requests are pending. |

To determine what operations are pending, choose the **Description** tab and check to see how many pending creations or deletions are shown. You cannot have both pending creations and pending deletions.

**Using the AWS CLI with ElastiCache**

1. Identify the IDs of the nodes that you want to remove. For more information, see Viewing an ElastiCache cluster's details.
2. Use the `decrease-replica-count` CLI operation with a list of the nodes to remove, as in the following example.

   To remove nodes from a cluster using the command-line interface, use the command `decrease-replica-count` with the following parameters:

   - `--replication-group-id` The ID of the replication group that you want to remove nodes from.
   - `--new-replica-count` The `--new-replica-count` parameter specifies the number of nodes that you want in this cluster after the modification is applied.

- `--replicas-to-remove` A list of node IDs that you want removed from this cluster.
- `--apply-immediately` or `--no-apply-immediately` Specifies whether to remove these nodes immediately or at the next maintenance window.
- `--region` Specifies the AWS Region of the cluster that you want to remove nodes from.

> **Note**
>
> You can pass only one of `--replicas-to-remove` or `--new-replica-count` parameters when calling this operation.

For Linux, macOS, or Unix:

```
aws elasticache decrease-replica-count \
    --replication-group-id my-replication-group \
    --new-replica-count 2 \
    --region us-east-2 \
    --apply-immediately
```

For Windows:

```
aws elasticache decrease-replica-count ^
    --replication-group-id my-replication-group ^
    --new-replica-count 3 ^
    --region us-east-2 ^
    --apply-immediately
```

This operation produces output similar to the following (JSON format):

```
{
    "ReplicationGroup": {
        "ReplicationGroupId": "node-test",
        "Description": "node-test"
       },
        "Status": "modifying",
        "PendingModifiedValues": {},
        "MemberClusters": [
            "node-test-001",
            "node-test-002",
```

```
            "node-test-003",
            "node-test-004",
            "node-test-005",
            "node-test-006"
        ],
        "NodeGroups": [
            {
                "NodeGroupId": "0001",
                "Status": "modifying",
                "PrimaryEndpoint": {
                    "Address": "node-test.zzzzzz.ng.0001.usw2.cache.amazonaws.com",
                    "Port": 6379
                },
                "ReaderEndpoint": {
                    "Address": "node-test-
ro.zzzzzz.ng.0001.usw2.cache.amazonaws.com",
                    "Port": 6379
                },
                "NodeGroupMembers": [
                    {
                        "CacheClusterId": "node-test-001",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-001.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2a",
                        "CurrentRole": "primary"
                    },
                    {
                        "CacheClusterId": "node-test-002",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-002.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2c",
                        "CurrentRole": "replica"
                    },
                    {
                        "CacheClusterId": "node-test-003",
                        "CacheNodeId": "0001",
```

```
                        "ReadEndpoint": {
                            "Address": "node-
test-003.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CurrentRole": "replica"
                    },
                    {
                        "CacheClusterId": "node-test-004",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-004.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2c",
                        "CurrentRole": "replica"
                    },
                    {
                        "CacheClusterId": "node-test-005",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-005.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CurrentRole": "replica"
                    },
                    {
                        "CacheClusterId": "node-test-006",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Address": "node-
test-006.zzzzzz.0001.usw2.cache.amazonaws.com",
                            "Port": 6379
                        },
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CurrentRole": "replica"
                    }
                ]
            }
        ],
```

```
        "SnapshottingClusterId": "node-test-002",
        "AutomaticFailover": "enabled",
        "MultiAZ": "enabled",
        "SnapshotRetentionLimit": 1,
        "SnapshotWindow": "07:30-08:30",
        "ClusterEnabled": false,
        "CacheNodeType": "cache.r5.large",
         "DataTiering": "disabled",
        "TransitEncryptionEnabled": false,
        "AtRestEncryptionEnabled": false,
        "ARN": "arn:aws:elasticache:us-west-2:123456789012:replicationgroup:node-
test"
    }
}
```

Alternatively, you could call `decrease-replica-count` and instead of passing in the `--new-replica-count` parameter, you could pass the `--replicas-to-remove` parameter, as shown following:

For Linux, macOS, or Unix:

```
aws elasticache decrease-replica-count \
    --replication-group-id my-replication-group \
    --replicas-to-remove node-test-003 \
    --region us-east-2 \
    --apply-immediately
```

For Windows:

```
aws elasticache decrease-replica-count ^
    --replication-group-id my-replication-group ^
    --replicas-to-remove node-test-003 ^
    --region us-east-2 ^
    --apply-immediately
```

For more information, see the AWS CLI topics `decrease-replica-count`.

**Using the ElastiCache API with Valkey or Redis OSS**

To remove nodes using the ElastiCache API, call the `DecreaseReplicaCount` API operation with the replication group Id and a list of nodes to remove, as shown:

- `ReplicationGroupId` The ID of the replication group that you want to remove nodes from.
- `ReplicasToRemove` The `ReplicasToRemove` parameter specifies the number of nodes that you want in this cluster after the modification is applied.
- `ApplyImmediately` Specifies whether to remove these nodes immediately or at the next maintenance window.
- `Region` Specifies the AWS Region of the cluster that you want to remove a node from.

The following example immediately removes nodes 0004 and 0005 from the cluster my-cluster.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DecreaseReplicaCount
    &ReplicationGroupId=my-replication-group
    &ApplyImmediately=true
    &ReplicasToRemove=node-test-003
    &Region us-east-2
    &Version=2014-12-01
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ElastiCache API topic `DecreaseReplicaCount`.

**Using the ElastiCache API with Memcached**

To remove nodes using the ElastiCache API, call the `ModifyCacheCluster` API operation with the cache cluster ID and a list of nodes to remove, as shown:

- `CacheClusterId` The ID of the cluster that you want to remove nodes from.
- `NumCacheNodes` The `NumCacheNodes` parameter specifies the number of nodes that you want in this cluster after the modification is applied.
- `CacheNodeIdsToRemove.member.n` The list of node IDs to remove from the cluster.
  - `CacheNodeIdsToRemove.member.1=0004`

  - `CacheNodeIdsToRemove.member.1=0005`
- `ApplyImmediately` Specifies whether to remove these nodes immediately or at the next maintenance window.
- `Region` Specifies the AWS Region of the cluster that you want to remove a node from.

The following example immediately removes nodes 0004 and 0005 from the cluster my-cluster.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyCacheCluster
    &CacheClusterId=my-cluster
    &ApplyImmediately=true
    &CacheNodeIdsToRemove.member.1=0004
    &CacheNodeIdsToRemove.member.2=0005
    &NumCacheNodes=3
    &Region us-east-2
    &Version=2014-12-01
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ElastiCache API topic `ModifyCacheCluster`.

# Canceling pending add or delete node operations in ElastiCache

If you elected to not apply an ElastiCache cluster change immediately, the operation has **pending** status until it is performed at your next maintenance window. You can cancel any pending add or delete operation.

To cancel a pending add or delete node operation with the AWS CLI, use the `modify-cache-cluster` command. Set `num-cache-nodes` equal to the current number of cache nodes in the cluster, and then add the `--apply-immediately` flag. This will override the pending change.

To cancel a pending node addition or deletion:

```
aws elasticache modify-cache-cluster
 --cache-cluster-id <your-cluster-id>
 --num-cache-nodes <current-number-of-nodes>
 --apply-immediately
```

If it isn't clear whether any node additions or deletions are pending, you can confirm their status with the `describe-cache-clusters` command:

```
aws elasticache describe-cache-clusters
 --cache-cluster-id <your-cluster-id>
```

Any pending nodes should appear in the `PendingModifiedValues`output. For example:

```
"PendingModifiedValues": {
 "NumCacheNodes": 3
 },
```

# Deleting a cluster in ElastiCache

As long as an ElastiCache cluster is in the *available* state, you are being charged for it, whether or not you are actively using it. To stop incurring charges, delete the cluster.

> **Warning**
>
> When you delete an ElastiCache cluster, your manual snapshots are retained. You can also create a final snapshot before the cluster is deleted. Automatic cache snapshots are not retained.

### Using the AWS Management Console

The following procedure deletes a single cluster from your deployment. To delete multiple clusters, repeat the procedure for each cluster that you want to delete. You do not need to wait for one cluster to finish deleting before starting the procedure to delete another cluster.

**To delete a cluster**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the ElastiCache engine dashboard, choose the engine that is running in the cluster that you want to delete.

   A list of all clusters running that engine appears.
3. To choose the cluster to delete, choose the cluster's name from the list of clusters.

   > **Important**
   >
   > You can only delete one cluster at a time from the ElastiCache console. Choosing multiple clusters disables the delete operation.
4. For **Actions**, choose **Delete**.
5. In the **Delete Cluster** confirmation screen, choose **Delete** to delete the cluster, or choose **Cancel** to keep the cluster.

   If you chose **Delete**, the status of the cluster changes to *deleting*.

As soon as your cluster is no longer listed in the list of clusters, you stop incurring charges for it.

### Using the AWS CLI to delete an ElastiCache cluster

The following code deletes the ElastiCache cluster `my-cluster`.

```
aws elasticache delete-cache-cluster --cache-cluster-id my-cluster
```

The `delete-cache-cluster` CLI action only deletes one cluster. To delete multiple clusters, call `delete-cache-cluster` for each cache cluster that you want to delete. You do not need to wait for one cluster to finish deleting before deleting another.

For Linux, macOS, or Unix:

```
aws elasticache delete-cache-cluster \
    --cache-cluster-id my-cluster \
    --region us-east-2
```

For Windows:

```
aws elasticache delete-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --region us-east-2
```

For more information, see the AWS CLI for ElastiCache topic `delete-cache-cluster`.

### Using the ElastiCache API

The following code deletes the cluster `my-cluster`.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DeleteCacheCluster
    &CacheClusterId=my-cluster
    &Region us-east-2
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20150202T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20150202T220302Z
```

```
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

The `DeleteCacheCluster` API operation only deletes one cluster. To delete multiple clusters, call `DeleteCacheCluster` for each cluster that you want to delete. You do not need to wait for one cluster to finish deleting before deleting another.

For more information, see the ElastiCache API reference topic `DeleteCacheCluster`.

# Accessing your ElastiCache cluster or replication group

Your Amazon ElastiCache instances are designed to be accessed through an Amazon EC2 instance.

If you launched your ElastiCache instance in an Amazon Virtual Private Cloud (Amazon VPC), you can access your ElastiCache instance from an Amazon EC2 instance in the same Amazon VPC. Or, by using VPC peering, you can access your ElastiCache instance from an Amazon EC2 in a different Amazon VPC.

If you launched your ElastiCache instance in EC2 Classic, you allow the EC2 instance to access your cluster by granting the Amazon EC2 security group associated with the instance access to your cache security group. By default, access to a cluster is restricted to the account that launched the cluster.

**Topics**


- Grant access to your clusteror replication group


## Grant access to your clusteror replication group

### You launched your cluster into EC2-VPC

If you launched your cluster into an Amazon Virtual Private Cloud (Amazon VPC), you can connect to your ElastiCache cluster only from an Amazon EC2 instance that is running in the same Amazon VPC. In this case, you will need to grant network ingress to the cluster.

> **Note**
>
> If your are using *Local Zones*, make sure you have enabled it. For more information, see Enable Local Zones. By doing so, your VPC is extended to that Local Zone and your VPC will treat the subnet as any subnet in any other Availability Zone and relevant gateways, route tables and other security group considerations. will be automatically adjusted.

**To grant network ingress from an Amazon VPC security group to a cluster**

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, under **Network & Security**, choose **Security Groups**.

3. From the list of security groups, choose the security group for your Amazon VPC. Unless you created a security group for ElastiCache use, this security group will be named *default*.
4. Choose the **Inbound** tab, and then do the following:

    a. Choose **Edit**.

    b. Choose **Add rule**.

    c. In the **Type** column, choose **Custom TCP rule**.

    d. In the **Port range** box, type the port number for your cluster node. This number must be the same one that you specified when you launched the cluster. The default port for Memcached is **11211** The default port for Valkey and Redis OSS is **6379**.

    e. In the **Source** box, choose **Anywhere** which has the port range (0.0.0.0/0) so that any Amazon EC2 instance that you launch within your Amazon VPC can connect to your ElastiCache nodes.

    > **Important**
    >
    > Opening up the ElastiCache cluster to 0.0.0.0/0 does not expose the cluster to the Internet because it has no public IP address and therefore cannot be accessed from outside the VPC. However, the default security group may be applied to other Amazon EC2 instances in the customer's account, and those instances may have a public IP address. If they happen to be running something on the default port, then that service could be exposed unintentionally. Therefore, we recommend creating a VPC Security Group that will be used exclusively by ElastiCache. For more information, see Custom Security Groups.

    f. Choose **Save**.

When you launch an Amazon EC2 instance into your Amazon VPC, that instance will be able to connect to your ElastiCache cluster.

### Accessing ElastiCache resources from outside AWS

Amazon ElastiCache is an AWS service that provides cloud-based in-memory key-value store. Node-based clusters and Serverless caches with a VPC endpoint are accessed from within AWS. Serverless caches with a public endpoint can be accessed directly over the internet. However, if the ElastiCache cluster is hosted inside a VPC, you can use an EC2 instance configured for Network Address Translation (NAT) with port forwarding to provide outside access.

> **Important**
>
> This approach should be used for testing and development purposes only. For production workloads requiring external access, consider using AWS Site-to-Site VPN or AWS Client VPN instead.

#### Requirements

The following requirements must be met for you to access your ElastiCache resources from outside AWS:

- The cluster must reside within a VPC and be accessed through a Network Address Translation (NAT) instance. There are no exceptions to this requirement.
- The port-forwarding EC2 instance must be launched in the same VPC as the cluster.
- The port-forwarding EC2 instance must be launched in a public subnet separate from the cluster.
- An Elastic IP Address (EIP) must be associated with the EC2 instance. The port forwarding feature of iptables is used to forward a port on the instance to the cache node port within the VPC.

#### Considerations

The following considerations should be kept in mind when accessing your ElastiCache resources from outside ElastiCache.

- Clients connect to the EIP and cache port of the NAT instance. Port forwarding on the NAT instance forwards traffic to the appropriate cluster node.
- If a cluster node is added or replaced, the iptables rules need to be updated to reflect this change.

### Limitations

This approach should be used for testing and development purposes only. It is not recommended for production use due to the following limitations:

- The NAT instance is acting as a proxy between clients and multiple clusters. The addition of a proxy impacts the performance of the cluster. The impact increases with number of clusters you are accessing through the NAT instance.
- The traffic from clients to the NAT instance is unencrypted. Therefore, you should avoid sending sensitive data via the NAT instance.
- The NAT instance adds the overhead of maintaining another instance.
- The NAT instance serves as a single point of failure. For high availability, consider using a NAT gateway instead.
- This approach does not work with Valkey or Redis OSS (cluster mode enabled). In cluster mode, clients follow `MOVED` and `ASK` redirects to the cluster's internal node addresses. These addresses are not reachable through the static port-forwarding rules (iptables) on the NAT instance. Use this approach only with Memcached or with cluster mode disabled.

**⚠ Security considerations for external access**

Port forwarding relays a cache port to a public Elastic IP address. If the NAT instance's security group allows access from a broad range of source addresses, this can expose your cache to the internet. Memcached provides no built-in authentication, and the NAT instance does not encrypt traffic in transit. Restrict the NAT instance security group to the specific trusted client IP addresses that require access. Do not send sensitive data through this connection.

This approach cannot provide encrypted, authenticated access to the cache. For in-transit encryption with Transport Layer Security (TLS), connect to an encryption-enabled cache from within the VPC. Alternatively, use AWS Site-to-Site VPN or AWS Client VPN for external access. For more information, see ElastiCache in-transit encryption (TLS).

### How to access ElastiCache resources from outside AWS

The following procedure demonstrates how to connect to your ElastiCache resources using a NAT instance.

These steps assume the following:

- `iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 6380 -j DNAT --to 10.0.1.231:6379`
- `iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 6381 -j DNAT --to 10.0.1.232:6379`

Next you need NAT in the opposite direction:

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 10.0.0.55
```

You also need to enable IP forwarding, which is disabled by default:

```
sudo sed -i 's/net.ipv4.ip_forward=0/net.ipv4.ip_forward=1/g' /etc/
sysctl.conf sudo sysctl --system
```

- You are accessing a Memcached cluster with:
  - IP address – *10.0.1.230*
  - Default Memcached port – *11211*
  - Security group – **10\.0\.0\.55**
- You are accessing a Valkey or Redis OSS cluster with:
  - IP address – *10.0.1.230*
  - Default port – *6379*
  - Security group – *sg-bd56b7da*
  - AWS instance IP address – *198.99.100.27*
- Your trusted client has the IP address *198.51.100.27*.
- Your NAT instance has the Elastic IP Address *203.0.113.73*.
- Your NAT instance has security group *sg-ce56b7a9*.

**To connect to your ElastiCache resources using a NAT instance**

1. Launch an EC2 instance in the same VPC as your cluster but in a public subnet. This instance will act as your NAT/port-forwarding host.

Use a current Amazon Linux 2023 AMI (or other Linux AMI of your choice). Select an instance size based on your expected traffic. The instance needs a public subnet, an Elastic IP address, and IP forwarding enabled.

For information about launching EC2 instances, see Launch an instance in the Amazon Elastic Compute Cloud User Guide.

2. Create security group rules for the cluster and NAT instance.

   The NAT instance security group and the cluster instance should have the following rules:

   - Two inbound rules
     - With Memcached, the first rule is to allow TCP connections from trusted clients to each cache port forwarded from the NAT instance (11211 - 11213).
     - With Valkey and Redis OSS, the first rule is to allow TCP connections from trusted clients to each cache port forwarded from the NAT instance (6379 - 6381).
     - A second rule to allow SSH access to trusted clients.

   **NAT instance security group - inbound rules with Memcached**

| Type | Protocol | Port range | Source |
|---|---|---|---|
| Custom TCP Rule | TCP | 11211-11213 | 198.51.100.27/32 |
| SSH | TCP | 22 | 198.51.100.27/32 |

   **NAT instance security group - inbound rules with Valkey or Redis OSS**

| Type | Protocol | Port range | Source |
|---|---|---|---|
| Custom TCP Rule | TCP | 6379-6380 | 198.51.100.27/32 |
| SSH | TCP | 22 | 203.0.113.73/32 |

   - With Memcached, an outbound rule to allow TCP connections to cache port (11211).

**NAT instance security group - outbound rule**

| Type | Protocol | Port range | Destination |
|---|---|---|---|
| Custom TCP Rule | TCP | 11211 | sg-ce56b7a9 (NAT Security Group) |

- With Valkey or Redis OSS, an outbound rule to allow TCP connections to cache port (6379).

  **NAT instance security group - outbound rule**

| Type | Protocol | Port range | Destination |
|---|---|---|---|
| Custom TCP Rule | TCP | 6379 | sg-ce56b7a9 (NAT Security Group) |

- With Memcached, an inbound rule for the cluster's security group that allows TCP connections from the NAT instance to the cache port (11211).

  **Cluster instance security group - inbound rule**

| Type | Protocol | Port range | Source |
|---|---|---|---|
| Custom TCP Rule | TCP | 11211 | sg-ce56b7a9 (NAT Security Group) |

- With Valkey or Redis OSS, an inbound rule for the cluster's security group that allows TCP connections from the NAT instance to the cache port (6379).

  **Cluster instance security group - inbound rule**

| Type | Protocol | Port range | Source |
|---|---|---|---|
| Custom TCP Rule | TCP | 6379 | sg-ce56b7a9 (NAT Security Group) |

3. Validate the rules.

   - Confirm that the trusted client is able to SSH to the NAT instance.
   - Confirm that the trusted client is able to connect to the cluster from the NAT instance.

4. **Memcached**

Add an iptables rule to the NAT instance.

An iptables rule must be added to the NAT table for each node in the cluster to forward the cache port from the NAT instance to the cluster node. An example might look like the following:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 11211 -j DNAT --to
 10.0.1.230:11211
```

The port number must be unique for each node in the cluster. For example, if working with a three node Memcached cluster using ports 11211 - 11213, the rules would look like the following:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 11211 -j DNAT --to
 10.0.1.230:11211
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 11212 -j DNAT --to
 10.0.1.231:11211
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 11213 -j DNAT --to
 10.0.1.232:11211
```

Confirm that the trusted client is able to connect to the cluster.

The trusted client should connect to the EIP associated with the NAT instance and the cluster port corresponding to the appropriate cluster node. For example, the connection string for PHP might look like the following:

```
$memcached->connect( '203.0.113.73', 11211 );
$memcached->connect( '203.0.113.73', 11212 );
$memcached->connect( '203.0.113.73', 11213 );
```

A telnet client can also be used to verify the connection. For example:

```
telnet 203.0.113.73 11211
telnet 203.0.113.73 11212
telnet 203.0.113.73 11213
```

**Valkey or Redis OSS**

Add an iptables rule to the NAT instance.

An iptables rule must be added to the NAT table for each node in the cluster to forward the cache port from the NAT instance to the cluster node. An example might look like the following:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 6379 -j DNAT --to
 10.0.1.230:6379
```

The port number must be unique for each node in the cluster. For example, if working with a three node Redis OSS cluster using ports 6379 - 6381, the rules would look like the following:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 6379 -j DNAT --to
 10.0.1.230:6379
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 6380 -j DNAT --to
 10.0.1.231:6379
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 6381 -j DNAT --to
 10.0.1.232:6379
```

Confirm that the trusted client is able to connect to the cluster.

The trusted client should connect to the EIP associated with the NAT instance and the cluster port corresponding to the appropriate cluster node. For example, the connection string for PHP might look like the following:

```
redis->connect( '203.0.113.73', 6379 );
redis->connect( '203.0.113.73', 6380 );
redis->connect( '203.0.113.73', 6381 );
```

A telnet client can also be used to verify the connection. For example:

```
telnet 203.0.113.73 6379
telnet 203.0.113.73 6380
telnet 203.0.113.73 6381
```

5. Save the iptables configuration.

   Save the rules after you test and verify them. If you are using a Redhat-based Linux distribution (like Amazon Linux), run the following command:

```
service iptables save
```

**Related topics**

The following topics may be of additional interest.

- Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC
- Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center
- Configuring ElastiCache Clients
- NAT Gateways

## Finding connection endpoints in ElastiCache

Your application connects to your ElastiCache cluster using endpoints. An endpoint is a node or cluster's unique address.

You can also establish a private connection between your VPC and ElastiCache API endpoints by creating an interface VPC endpoint through AWS PrivateLink. For more information, see ElastiCache API and interface VPC endpoints (AWS PrivateLink).

**Which endpoints to use with Valkey or Redis OSS.**

- For a **standalone node**, use the node's endpoint for both read and write operations.

- For **Valkey or Redis OSS (cluster mode disabled) clusters**, use the *Primary Endpoint* for all write operations. Use the *Reader Endpoint* to evenly split incoming connections to the endpoint between all read replicas. Use the individual *Node Endpoints* for read operations (In the API/CLI these are referred to as Read Endpoints).

- For **Valkey or Redis OSS (cluster mode enabled) clusters**, use the cluster's *Configuration Endpoint* for all operations that support cluster mode enabled commands. You must use a client that supports either Valkey Cluster or Redis OSS Cluster. You can still read from individual node endpoints (In the API/CLI these are referred to as Read Endpoints).

The following sections guide you through discovering the endpoints you'll need for the engine you're running.

**Which endpoints to use with Memcached.**

For **ElastiCache serverless cache for Memcached**, simply acquire the cluster endpoint DNS and port from the console.

From the AWS CLI, use the `describe-serverless-caches` command to acquire the Endpoint information.

Linux

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

Windows

```
aws elasticache describe-serverless-caches --serverless-cache-name CacheName
```

The output from the above operation should look something like this (JSON format):

```
{
    "ServerlessCaches": [
        {
            "ServerlessCacheName": "serverless-memcached",
            "Description": "test",
            "CreateTime": 1697659642.136,
            "Status": "available",
            "Engine": "memcached",
            "MajorEngineVersion": "1.6",
            "FullEngineVersion": "21",
            "SecurityGroupIds": [
                "sg-083eda453e1e51310"
            ],
            "Endpoint": {
                "Address": "serverless-memcached-01.amazonaws.com",
                "Port":11211
            },
            "ARN": "<the ARN>",
            "SubnetIds": [
                "subnet-0cf759df15bd4dc65",
                "subnet-09e1307e8f1560d17"
```

```
            ],
            "SnapshotRetentionLimit": 0,
            "DailySnapshotTime": "03:00"
        }
    ]
}
```

For an **instance based Memcached cluster**, if you use Automatic Discovery then you can use the cluster's *configuration endpoint* to configure your Memcached client. This means you must use a client that supports Automatic Discovery.

If you don't use Automatic Discovery, you must configure your client to use the individual node endpoints for reads and writes. You must also keep track of them as you add and remove nodes.

### Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)

If a Valkey or Redis OSS (cluster mode disabled) cluster has only one node, the node's endpoint is used for both reads and writes. If a Valkey or Redis OSS (cluster mode disabled) cluster has multiple nodes, there are three types of endpoints; the *primary endpoint*, the *reader endpoint* and the *node endpoints*.

The primary endpoint is a DNS name that always resolves to the primary node in the cluster. The primary endpoint is immune to changes to your cluster, such as promoting a read replica to the primary role. For write activity, we recommend that your applications connect to the primary endpoint.

A reader endpoint will evenly split incoming connections to the endpoint between all read replicas in a ElastiCache for Redis OSS cluster. Additional factors such as when the application creates the connections or how the application (re)-uses the connections will determine the traffic distribution. Reader endpoints keep up with cluster changes in real-time as replicas are added or removed. You can place your ElastiCache for Redis OSS cluster's multiple read replicas in different AWS Availability Zones (AZ) to ensure high availability of reader endpoints.

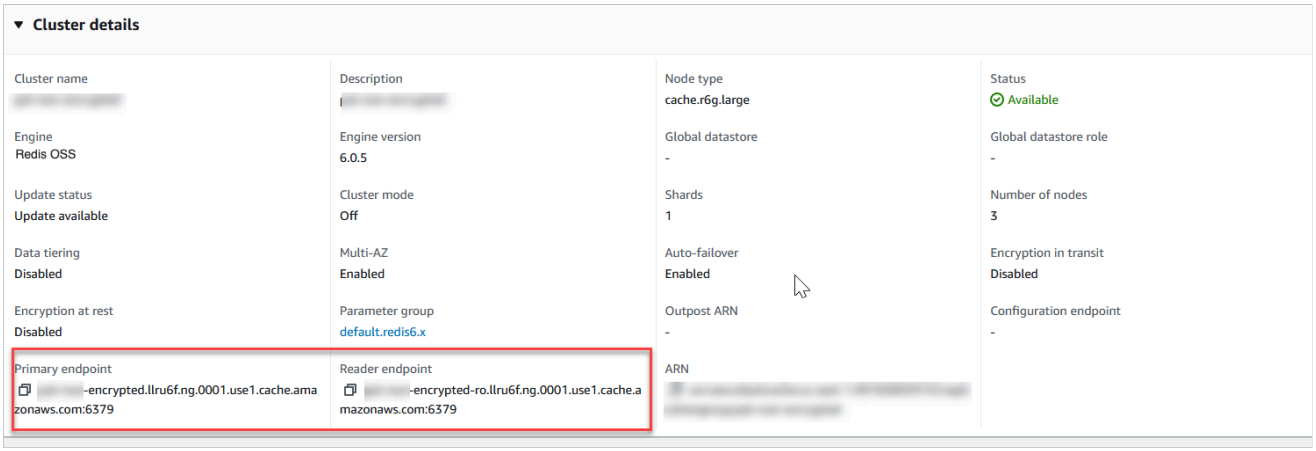
> **Note**
>
> A reader endpoint is not a load balancer. It is a DNS record that will resolve to an IP address of one of the replica nodes in a round robin fashion.

For read activity, applications can also connect to any node in the cluster. Unlike the primary endpoint, node endpoints resolve to specific endpoints. If you make a change in your cluster, such as adding or deleting a replica, you must update the node endpoints in your application.

**To find a Valkey or Redis OSS (cluster mode disabled) cluster's endpoints**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.

   The clusters screen will appear with a list of Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled) clusters.
3. To find the cluster's Primary and/or Reader endpoints, choose the cluster's name (not the button to its left).

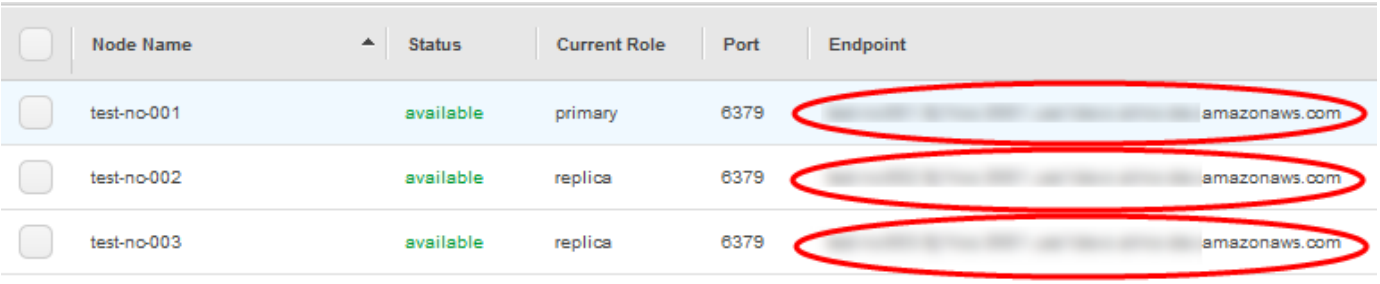



*Primary and Reader endpoints for a Valkey or Redis OSS (cluster mode disabled) cluster*

If there is only one node in the cluster, there is no primary endpoint and you can continue at the next step.

4. If the Valkey or Redis OSS (cluster mode disabled) cluster has replica nodes, you can find the cluster's replica node endpoints by choosing the cluster's name and then choosing the **Nodes** tab.

   The nodes screen appears with each node in the cluster, primary and replicas, listed with its endpoint.




*Node endpoints for a Valkey or Redis OSS (cluster mode disabled) cluster*

5. To copy an endpoint to your clipboard:

   a. One endpoint at a time, find the endpoint you want to copy.

   b. Choose the copy icon directly in front of the endpoint.

The endpoint is now copied to your clipboard. For information on using the endpoint to connect to a node, see Connecting to Memcached nodes.

A Valkey or Redis OSS (cluster mode disabled) primary endpoint looks something like the following. There is a difference depending upon whether or not In-Transit encryption is enabled.

**In-transit encryption not enabled**

```
clusterName.xxxxxx.nodeId.regionAndAz.cache.amazonaws.com:port

redis-01.7abc2d.0001.usw2.cache.amazonaws.com:6379
```

**In-transit encryption enabled**

```
master.clusterName.xxxxxx.regionAndAz.cache.amazonaws.com:port

master.ncit.ameaqx.use1.cache.amazonaws.com:6379
```

**Finding Endpoints for a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (Console)**

A Valkey or Redis OSS (cluster mode enabled) cluster has a single configuration endpoint. By connecting to the configuration endpoint, your application is able to discover the primary and read endpoints for each shard in the cluster.

**To find a Valkey or Redis OSS (cluster mode enabled) cluster's endpoint**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.

   The clusters screen will appear with a list of clusters. Choose the cluster you wish to connect to.
3. To find the cluster's Configuration endpoint, choose the cluster's name (not the radio button).
4. The **Configuration endpoint** is displayed under **Cluster details**. To copy it, choose the *copy* icon to the left of the endpoint.

### Finding a Cluster's Endpoints (Console) (Memcached)

All Memcached endpoints are read/write endpoints. To connect to nodes in a Memcached cluster your application can use either the endpoints for each node, or the cluster's configuration endpoint along with Automatic Discovery. To use Automatic Discovery you must use a client that supports Automatic Discovery.

When using Automatic Discovery, your client application connects to your Memcached cluster using the configuration endpoint. As you scale your cluster by adding or removing nodes, your application will automatically "know" all the nodes in the cluster and be able to connect to any of them. Without Automatic Discovery your application would have to do this, or you'd have to manually update endpoints in your application each time you added or removed a node.

To copy an endpoint, choose the copy icon directly in front of the endpoint address. For information on using the endpoint to connect to a node, see Connecting to Memcached nodes.

Configuration and node endpoints look very similar. The differences are highlighted with **bold** following.

```
myclustername.xxxxxx.cfg.usw2.cache.amazonaws.com:port   # configuration endpoint
 contains "cfg"
myclustername.xxxxxx.0001.usw2.cache.amazonaws.com:port  # node endpoint for node 0001
```

> **Important**
>
> If you choose to create a CNAME for your Memcached configuration endpoint, in order for your automatic discovery client to recognize the CNAME as a configuration endpoint, you must include `.cfg.` in the CNAME.

### Finding Endpoints (AWS CLI)

For Memcached, you can use the AWS CLI for Amazon ElastiCache to discover the endpoints for nodes and clusters.

For Redis OSS, you can use the AWS CLI for Amazon ElastiCache to discover the endpoints for nodes, clusters, and also replication groups.

#### Topics

- Finding Endpoints for Nodes and Clusters (AWS CLI)
- Finding the Endpoints for Valkey or Redis OSS Replication Groups (AWS CLI)

### Finding Endpoints for Nodes and Clusters (AWS CLI)

You can use the AWS CLI to discover the endpoints for a cluster and its nodes with the `describe-cache-clusters` command. For Valkey or Redis OSS clusters, the command returns the cluster endpoint.  For Memcached clusters, the command returns the configuration endpoint. If you include the optional parameter `--show-cache-node-info`, the command will also return the endpoints of the individual nodes in the cluster.

#### Example

The following command retrieves the configuration endpoint (`ConfigurationEndpoint`) and individual node endpoints (`Endpoint`) for the Memcached cluster *mycluster*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-clusters \
    --cache-cluster-id mycluster \
    --show-cache-node-info
```

For Windows:

```
aws elasticache describe-cache-clusters ^
    --cache-cluster-id mycluster ^
    --show-cache-node-info
```

Output from the above operation should look something like this (JSON format).

```
{
```

```
    "CacheClusters": [
    {
        "Engine": "memcached",
        "CacheNodes": [
           {
              "CacheNodeId": "0001",
              "Endpoint": {
                 "Port": 11211,
                 "Address": "mycluster.amazonaws.com"
              },
                 "CacheNodeStatus": "available",
                 "ParameterGroupStatus": "in-sync",
                 "CacheNodeCreateTime": "2016-09-22T21:30:29.967Z",
                 "CustomerAvailabilityZone": "us-west-2b"
           },
           {
              "CacheNodeId": "0002",
              "Endpoint": {
                 "Port": 11211,
                 "Address": "mycluster.amazonaws.com"
              },
                 "CacheNodeStatus": "available",
                 "ParameterGroupStatus": "in-sync",
                 "CacheNodeCreateTime": "2016-09-22T21:30:29.967Z",
                 "CustomerAvailabilityZone": "us-west-2b"
           },
           {
                 "CacheNodeId": "0003",
                 "Endpoint": {
                    "Port": 11211,
                    "Address": "mycluster.amazonaws.com"
                 },
                    "CacheNodeStatus": "available",
                    "ParameterGroupStatus": "in-sync",
                    "CacheNodeCreateTime": "2016-09-22T21:30:29.967Z",
                    "CustomerAvailabilityZone": "us-west-2b"
           }
        ],
        "CacheParameterGroup": {
        "CacheNodeIdsToReboot": [],
        "CacheParameterGroupName": "default.memcached1.4",
        "ParameterApplyStatus": "in-sync"
             },
             "CacheClusterId": "mycluster",
```

```
            "PreferredAvailabilityZone": "us-west-2b",
            "ConfigurationEndpoint": {
                "Port": 11211,
                "Address": "mycluster.amazonaws.com"
            },
            "CacheSecurityGroups": [],
            "CacheClusterCreateTime": "2016-09-22T21:30:29.967Z",
            "AutoMinorVersionUpgrade": true,
            "CacheClusterStatus": "available",
            "NumCacheNodes": 3,
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "CacheSubnetGroupName": "default",
            "EngineVersion": "1.4.24",
            "PendingModifiedValues": {},
            "PreferredMaintenanceWindow": "mon:09:00-mon:10:00",
            "CacheNodeType": "cache.m4.large",
             "DataTiering": "disabled"
        }
    ]
}
```

**Important**

If you choose to create a CNAME for your Memcached configuration endpoint, in order for your auto discovery client to recognize the CNAME as a configuration endpoint, you must include `.cfg.` in the CNAME. For example, `mycluster.cfg.local` in your php.ini file for the `session.save_path` parameter.

### Example

For Valkey and Redis OSS, the following command retrieves the cluster information for the single-node cluster *mycluster*.

**Important**

The parameter `--cache-cluster-id` can be used with single-node Valkey or Redis OSS (cluster mode disabled) cluster id or specific node ids in replication groups. The `--cache-cluster-id` of a replication group is a 4-digit value such as `0001`. If `--cache-cluster-`

`id` is the id of a cluster (node) in a replication group, the `replication-group-id` is included in the output.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-clusters \
    --cache-cluster-id redis-cluster \
    --show-cache-node-info
```

For Windows:

```
aws elasticache describe-cache-clusters ^
    --cache-cluster-id redis-cluster ^
    --show-cache-node-info
```

Output from the above operation should look something like this (JSON format).

```
{
    "CacheClusters": [
        {
            "CacheClusterStatus": "available",
            "SecurityGroups": [
                {
                    "SecurityGroupId": "sg-77186e0d",
                    "Status": "active"
                }
            ],
            "CacheNodes": [
                {
                    "CustomerAvailabilityZone": "us-east-1b",
                    "CacheNodeCreateTime": "2018-04-25T18:19:28.241Z",
                    "CacheNodeStatus": "available",
                    "CacheNodeId": "0001",
                    "Endpoint": {
                        "Address": "redis-cluster.amazonaws.com",
                        "Port": 6379
                    },
                    "ParameterGroupStatus": "in-sync"
                }
            ],
            "AtRestEncryptionEnabled": false,
```

```
            "CacheClusterId": "redis-cluster",
            "TransitEncryptionEnabled": false,
            "CacheParameterGroup": {
                "ParameterApplyStatus": "in-sync",
                "CacheNodeIdsToReboot": [],
                "CacheParameterGroupName": "default.redis7"
            },
            "NumCacheNodes": 1,
            "PreferredAvailabilityZone": "us-east-1b",
            "AutoMinorVersionUpgrade": true,
            "Engine": "redis",
            "AuthTokenEnabled": false,
            "PendingModifiedValues": {},
            "PreferredMaintenanceWindow": "tue:08:30-tue:09:30",
            "CacheSecurityGroups": [],
            "CacheSubnetGroupName": "default",
            "CacheNodeType": "cache.t2.small",
             "DataTiering": "disabled"
            "EngineVersion": "7.1",
            "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
            "CacheClusterCreateTime": "2018-04-25T18:19:28.241Z"
        }
    ]
}
```

For more information, see the topic describe-cache-clusters.

**Finding the Endpoints for Valkey or Redis OSS Replication Groups (AWS CLI)**

You can use the AWS CLI to discover the endpoints for a replication group and its clusters with the `describe-replication-groups` command. The command returns the replication group's primary endpoint and a list of all the clusters (nodes) in the replication group with their endpoints, along with the reader endpoint.

The following operation retrieves the primary endpoint and reader endpoint for the replication group `myreplgroup`. Use the primary endpoint for all write operations.

```
aws elasticache describe-replication-groups \
    --replication-group-id myreplgroup
```

For Windows:

```
aws elasticache describe-replication-groups ^
    --replication-group-id myreplgroup
```

Output from this operation should look something like this (JSON format).

```
{
   "ReplicationGroups": [
     {
       "Status": "available",
       "Description": "test",
       "NodeGroups": [
         {
            "Status": "available",
               "NodeGroupMembers": [
                  {
                     "CurrentRole": "primary",
                     "PreferredAvailabilityZone": "us-west-2a",
                     "CacheNodeId": "0001",
                     "ReadEndpoint": {
                        "Port": 6379,
                        "Address": "myreplgroup-001.amazonaws.com"
                     },
                     "CacheClusterId": "myreplgroup-001"
                  },
                  {
                     "CurrentRole": "replica",
                     "PreferredAvailabilityZone": "us-west-2b",
                     "CacheNodeId": "0001",
                     "ReadEndpoint": {
                        "Port": 6379,
                        "Address": "myreplgroup-002.amazonaws.com"
                     },
                     "CacheClusterId": "myreplgroup-002"
                  },
                  {
                     "CurrentRole": "replica",
                     "PreferredAvailabilityZone": "us-west-2c",
                     "CacheNodeId": "0001",
                     "ReadEndpoint": {
                        "Port": 6379,
                        "Address": "myreplgroup-003.amazonaws.com"
                     },
                     "CacheClusterId": "myreplgroup-003"
```

```
                }
            ],
            "NodeGroupId": "0001",
            "PrimaryEndpoint": {
               "Port": 6379,
               "Address": "myreplgroup.amazonaws.com"
            },
            "ReaderEndpoint": {
               "Port": 6379,
               "Address": "myreplgroup-ro.amazonaws.com"
            }
         }
      ],
      "ReplicationGroupId": "myreplgroup",
      "AutomaticFailover": "enabled",
      "SnapshottingClusterId": "myreplgroup-002",
      "MemberClusters": [
         "myreplgroup-001",
         "myreplgroup-002",
         "myreplgroup-003"
      ],
      "PendingModifiedValues": {}
   }
  ]
}
```

For more information, see describe-replication-groups in the *AWS CLI Command Reference*.

### Finding Endpoints (ElastiCache API)

For Memcached, you can use the Amazon ElastiCache API to discover the endpoints for nodes and clusters.

For Redis OSS, you can use the Amazon ElastiCache API to discover the endpoints for nodes, clusters, and also replication groups.

#### Topics

- Finding Endpoints for Nodes and Clusters (ElastiCache API)
- Finding Endpoints for Valkey or Redis OSS Replication Groups (ElastiCache API)

### Finding Endpoints for Nodes and Clusters (ElastiCache API)

You can use the ElastiCache API to discover the endpoints for a cluster and its nodes with the `DescribeCacheClusters` action. For Valkey or Redis OSS clusters, the command returns the cluster endpoint. For Memcached clusters, the command returns the configuration endpoint. If you include the optional parameter `ShowCacheNodeInfo`, the action also returns the endpoints of the individual nodes in the cluster.

#### Example

For Memcached, the following command retrieves the configuration endpoint (`ConfigurationEndpoint`) and individual node endpoints (`Endpoint`) for the Memcached cluster *mycluster*.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeCacheClusters
    &CacheClusterId=mycluster
    &ShowCacheNodeInfo=true
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &Version=2015-02-02
    &X-Amz-Credential=<credential>
```

**Important**

If you choose to create a CNAME for your Memcached configuration endpoint, in order for your auto discovery client to recognize the CNAME as a configuration endpoint, you must

include `.cfg.` in the CNAME. For example, `mycluster.cfg.local` in your php.ini file for the `session.save_path` parameter.

#### Finding Endpoints for Valkey or Redis OSS Replication Groups (ElastiCache API)

You can use the ElastiCache API to discover the endpoints for a replication group and its clusters with the `DescribeReplicationGroups` action. The action returns the replication group's primary endpoint and a list of all the clusters in the replication group with their endpoints, along with the reader endpoint.

The following operation retrieves the primary endpoint (PrimaryEndpoint), reader endpoint (ReaderEndpoint) and individual node endpoints (ReadEndpoint) for the replication group `myreplgroup`. Use the primary endpoint for all write operations.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeReplicationGroups
    &ReplicationGroupId=myreplgroup
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &Version=2015-02-02
    &X-Amz-Credential=<credential>
```

For more information, see DescribeReplicationGroups.

## Working with shards in ElastiCache

A shard (API/CLI: node group) is a collection of one to six ElastiCache for Valkey or Redis OSS nodes. A Valkey or Redis OSS (cluster mode disabled) cluster will never have more than one shard. With shards, you can separate large databases into smaller, faster, and more easily managed parts called data shards. This can increase database efficiency by distributing operations across multiple separate sections. Using shards can offer many benefits including improved performance, scalability, and cost efficiency.

You can create a cluster with higher number of shards and lower number of replicas totaling up to 90 nodes per cluster. This cluster configuration can range from 90 shards and 0 replicas to 15 shards and 5 replicas, which is the maximum number of replicas allowed. The cluster's data is partitioned across the cluster's shards. If there is more than one node in a shard, the shard

implements replication with one node being the read/write primary node and the other nodes read-only replica nodes.

The node or shard limit can be increased to a maximum of 500 per cluster if the engine version is Valkey 7.2 and higher, or Redis OSS 5.0.6 to 7.1. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group.

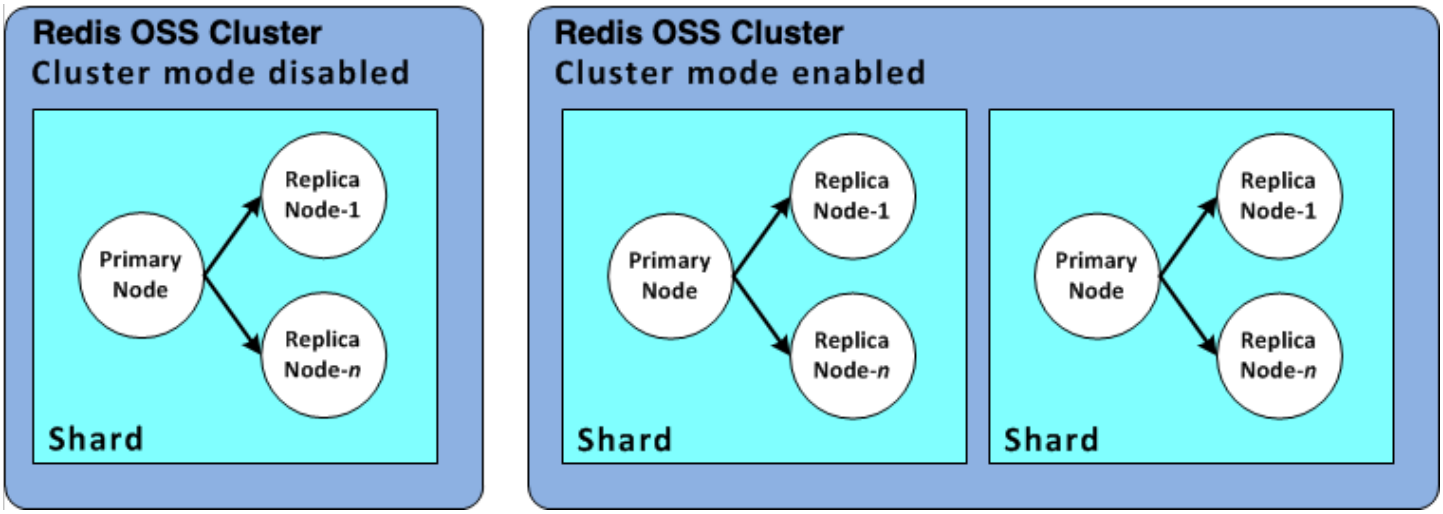
For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS service limits and choose the limit type **Nodes per cluster per instance type**.

When you create a Valkey or Redis OSS (cluster mode enabled) cluster using the ElastiCache console, you specify the number of shards in the cluster and the number of nodes in the shards. For more information, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console). If you use the ElastiCache API or AWS CLI to create a cluster (called *replication group* in the API/CLI), you can configure the number of nodes in a shard (API/CLI: node group) independently. For more information, see the following:

- API: CreateReplicationGroup
- CLI: create-replication-group

Each node in a shard has the same compute, storage and memory specifications. The ElastiCache API lets you control shard-wide attributes, such as the number of nodes, security settings, and system maintenance windows.




*Valkey or Redis OSS shard configurations*

For more information, see Offline resharding for Valkey or Redis OSS (cluster mode enabled) and Online resharding for Valkey or Redis OSS (cluster mode enabled).

## Finding a shard's ID

You can find a shard's ID using the AWS Management Console, the AWS CLI or the ElastiCache API.

### Using the AWS Management Console

### Topics

- For Valkey or Redis OSS (Cluster Mode Disabled)
- For Valkey or Redis OSS (Cluster Mode Enabled)

### For Valkey or Redis OSS (Cluster Mode Disabled)

Valkey or Redis OSS (cluster mode disabled) replication group shard IDs are always `0001`.

### For Valkey or Redis OSS (Cluster Mode Enabled)

The following procedure uses the AWS Management Console to find a Valkey or Redis OSS (cluster mode enabled)'s replication group's shard ID.

### To find the shard ID in a Valkey or Redis OSS (cluster mode enabled) replication group

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. On the navigation pane, choose **Valkey** or **Redis OSS**, then choose the name of the Valkey or Redis OSS (cluster mode enabled) replication group you want to find the shard IDs for.
3. In the **Shard Name** column, the shard ID is the last four digits of the shard name.

### Using the AWS CLI

To find shard (node group) ids for either Valkey or Redis OSS (cluster mode disabled) or Valkey or Redis OSS (cluster mode enabled) replication groups use the AWS CLI operation `describe-replication-groups` with the following optional parameter.

- **`--replication-group-id`**—An optional parameter which when used limits the output to the details of the specified replication group. If this parameter is omitted, the details of up to 100 replication groups is returned.

### Example

This command returns the details for `sample-repl-group`.

For Linux, macOS, or Unix:

```
aws elasticache describe-replication-groups \
    --replication-group-id sample-repl-group
```

For Windows:

```
aws elasticache describe-replication-groups ^
    --replication-group-id sample-repl-group
```

Output from this command looks something like this. The shard (node group) ids are *highlighted* here to make finding them easier.

```
{
    "ReplicationGroups": [
        {
            "Status": "available",
            "Description": "2 shards, 2 nodes (1 + 1 replica)",
            "NodeGroups": [
                {
                    "Status": "available",
                    "Slots": "0-8191",
```

```
                    "NodeGroupId": "0001",
                    "NodeGroupMembers": [
                        {
                            "PreferredAvailabilityZone": "us-west-2c",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "sample-repl-group-0001-001"
                        },
                        {
                            "PreferredAvailabilityZone": "us-west-2a",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "sample-repl-group-0001-002"
                        }
                    ]
                },
                {
                    "Status": "available",
                    "Slots": "8192-16383",
                    "NodeGroupId": "0002",
                    "NodeGroupMembers": [
                        {
                            "PreferredAvailabilityZone": "us-west-2b",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "sample-repl-group-0002-001"
                        },
                        {
                            "PreferredAvailabilityZone": "us-west-2a",
                            "CacheNodeId": "0001",
                            "CacheClusterId": "sample-repl-group-0002-002"
                        }
                    ]
                }
            ],
            "ConfigurationEndpoint": {
                "Port": 6379,
                "Address": "sample-repl-
group.9dcv5r.clustercfg.usw2.cache.amazonaws.com"
            },
            "ClusterEnabled": true,
            "ReplicationGroupId": "sample-repl-group",
            "SnapshotRetentionLimit": 1,
            "AutomaticFailover": "enabled",
            "SnapshotWindow": "13:00-14:00",
            "MemberClusters": [
                "sample-repl-group-0001-001",
```

```
                "sample-repl-group-0001-002",
                "sample-repl-group-0002-001",
                "sample-repl-group-0002-002"
            ],
            "CacheNodeType": "cache.m5.large",
            "DataTiering": "disabled",
            "PendingModifiedValues": {}
        }
    ]
}
```

### Using the ElastiCache API

To find shard (node group) ids for either Valkey or Redis OSS (cluster mode disabled) or Valkey or Redis OSS (cluster mode enabled) replication groups use the AWS CLI operation `describe-replication-groups` with the following optional parameter.

- **`ReplicationGroupId`**—An optional parameter which when used limits the output to the details of the specified replication group. If this parameter is omitted, the details of up to *`xxx`* replication groups is returned.

### Example

This command returns the details for `sample-repl-group`.

For Linux, macOS, or Unix:

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeReplicationGroup
   &ReplicationGroupId=sample-repl-group
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

# Comparing node-based Valkey, Memcached, and Redis OSS clusters

Amazon ElastiCache supports the Valkey, Memcached, and Redis OSS cache engines. Each engine provides some advantages. Use the information in this topic to help you choose the engine and version that best meets your requirements.

> **Important**
>
> After you create a cache, node-based cluster or replication group, you can upgrade to a newer engine version, but you cannot downgrade to an older engine version. If you want to use an older engine version, you must delete the existing cache, node-based cluster or replication group and create it again with the earlier engine version.

On the surface, the engines look similar. Each of them is an in-memory key-value store. However, in practice there are significant differences.

**Choose Memcached if the following apply for you:**

- You need the simplest model possible.
- You need to run large nodes with multiple cores or threads.
- You need the ability to scale out and in, adding and removing nodes as demand on your system increases and decreases.
- You need to cache objects.

**Choose Valkey or Redis OSS with ElastiCache if the following apply for you:**

- **ElastiCache version 9.0 for Valkey**

  You want built-in full-text search, aggregations, and hybrid search capabilities that extend the vector search introduced in Valkey 8.2, hash field expiration for per-field TTLs, multi-database support in cluster mode, polygon-based geospatial queries, up to 40% higher throughput with pipelining, or atomic slot migration for safer cluster scaling, or *durability* via a Multi-AZ transactional log for zero data loss during failures. For more information, see Valkey 9.0.
- **ElastiCache version 8.2 for Valkey**

You want native vector search with microsecond latency and 95%+ recall rate. For more information, see Valkey 8.2.

- **ElastiCache version 8.1 for Valkey**

  You want a new memory-efficient hash table that reduces memory overhead by up to 20%, native Bloom filter support, the COMMANDLOG command, and up to 514% higher throughput for BITCOUNT. For more information, see Valkey 8.1.

- **ElastiCache version 8.0 for Valkey**

  You want up to 20% better memory efficiency through embedded keys, per-slot metrics, dual-channel replication, and async I/O threading. For more information, see Valkey 8.0.

- **ElastiCache version 7.2 for Valkey or version 7.0 (Enhanced) for Redis OSS**

  You want to use Functions, Sharded Pub/Sub, or ACL improvements. For more information, see Redis OSS Version 7.0 (Enhanced).

- **ElastiCache version 6.2 (Enhanced) for Redis OSS**

  You want the ability to tier data between memory and SSD using the r6gd node type. For more information, see Data tiering.

- **ElastiCache version 6.0 (Enhanced) for Redis OSS**

  You want to authenticate users with role-based access control.

  For more information, see Redis OSS Version 6.0 (Enhanced).

**Comparison summary of Memcached, Valkey or Redis OSS (cluster mode disabled), and Valkey or Redis OSS (cluster mode enabled)**

| | Memcached | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|---|
| Engine versions+ | 1.4.5 and later | 4.0.10 and later | 4.0.10 and later |
| Data types | Simple | 2.8.x - Complex *<br>Complex | 3.2.x and later - Complex |
| Data partitioning | Yes | No | Yes |
| Cluster is modifiable | Yes | Yes | Yes - Limited |
| Online resharding | No | No | Yes |
| Encryption | in-transit 1.6.12 and later | 4.0.10 and later | 4.0.10 and later |
| Data tiering | No | 6.2 and later | 6.2 and later |
| Memory efficiencies | No | Valkey 8.0 and later | Valkey 8.0 and later |
| Bloom filters | No | Valkey 8.1 and later | Valkey 8.1 and later |
| Vector search | No | Valkey 8.2 and later | Valkey 8.2 and later |
| Full-text search | No | Valkey 9.0 and later | Valkey 9.0 and later |
| Hybrid search (text + vector) | No | Valkey 9.0 and later | Valkey 9.0 and later |
| Aggregation pipelines | No | Valkey 9.0 and later | Valkey 9.0 and later |
| Hash field expiration | No | Valkey 9.0 and later | Valkey 9.0 and later |
| Numbered databases in cluster mode | No | N/A | Valkey 9.0 and later |

| | Memcached | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|---|
| Durability | No | No | Valkey 9.0 and later (sync/async) |
| Compliance certifications | | | |
| Compliance Certification | Yes - 1.6.12 and later | 4.0.10 and later | 4.0.10 and later |
| FedRAMP | Yes - 1.6.12 and later | 4.0.10 and later | 4.0.10 and later |
| HIPAA | Yes | 4.0.10 and later | 4.0.10 and later |
| PCI DSS | | | |
| Multi-threaded | Yes | No | No |
| Node type upgrade | Yes (engine version 1.5+) | Yes | Yes |
| Engine upgrading | Yes | Yes | Yes |
| High availability (replication) | No | Yes | Yes |
| Automatic failover | No | Optional | Required |
| Pub/Sub capabilities | No | Yes | Yes |
| Sorted sets | No | Yes | Yes |
| Backup and restore | For serverless caches only, not applicable to node-based clusters | Yes | Yes |
| Geospatial indexing | No | 4.0.10 and later | Yes |

| | Memcached | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|---|

**Notes:**

string, objects (like databases)

* string, sets, sorted sets, lists, hashes, bitmaps, hyperloglog

string, sets, sorted sets, lists, hashes, bitmaps, hyperloglog, geospatial indexes

+ Excludes versions which are deprecated, have reached or soon to reach end of life.

After you choose the engine for your cluster, we recommend that you use the most recent version of that engine. For more information, see Supported node types.

# Choosing regions and availability zones for ElastiCache

You can provide additional scalability and reliability to your ElastiCache clusters by designating Regions and Availability Zones using the corresponding endpoint.

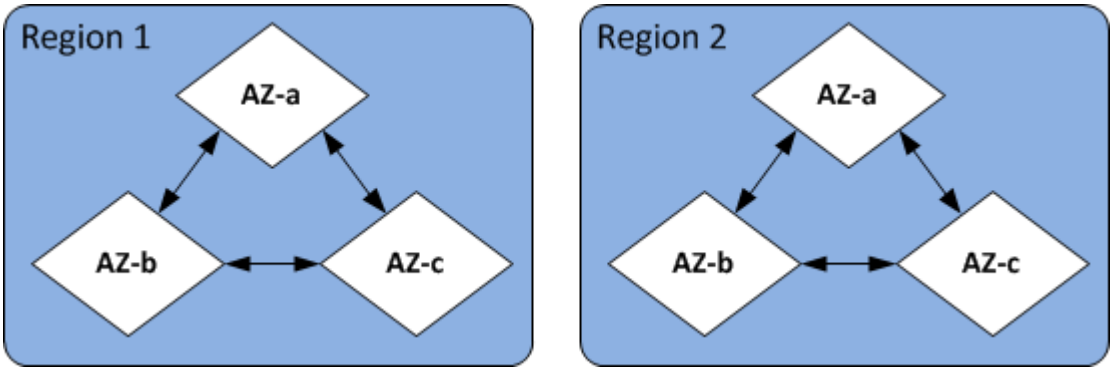
AWS Cloud computing resources are housed in highly available data center facilities. To provide additional scalability and reliability, these data center facilities are located in different physical locations. These locations are categorized by *regions* and *Availability Zones*.

AWS Regions are large and widely dispersed into separate geographic locations. Availability Zones are distinct locations within an AWS Region that are engineered to be isolated from failures in other Availability Zones. They provide inexpensive, low-latency network connectivity to other Availability Zones in the same AWS Region.

**Important**

Each region is completely independent. Any ElastiCache activity you initiate (for example, creating clusters) runs only in your current default region.

To create or work with a cluster in a specific region, use the corresponding regional service endpoint. For service endpoints, see Supported Regions & endpoints.




*Regions and Availability Zones*

**Topics**


- Availability Zone considerations with Memcached
- Locating your nodes
- Supported Regions & endpoints
- Using local zones with ElastiCache
- Using Outposts with ElastiCache


## Availability Zone considerations with Memcached

Distributing your Memcached nodes over multiple Availability Zones within a region helps protect you from the impact of a catastrophic failure, such as a power loss within an Availability Zone.

### Serverless Caching

ElastiCache serverless caching creates a highly available cache that spans multiple Availability Zones. You can specify subnets from different availability zones and same VPC as you create your serverless cluster or ElastiCache will choose subnets automatically from your default VPC.

### Designing your own ElastiCache for Memcached cluster

A Memcached cluster can have up to 300 nodes. When you create or add nodes to your Memcached cluster, you can specify a single Availability Zone for all your nodes, allow ElastiCache to choose a single Availability Zone for all your nodes, specify the Availability Zones for each node, or allow ElastiCache to choose an Availability Zone for each node. New nodes can be created in different Availability Zones as you add them to an existing Memcached cluster. Once a cache node is created, its Availability Zone cannot be modified.

If you want a cluster in a single Availability Zone cluster to have its nodes distributed across multiple Availability Zones, ElastiCache can create new nodes in the various Availability Zones. You can then delete some or all of the original cache nodes. We recommend this approach.

**To migrate Memcached nodes from a single Availability Zone to multiple availability zones**

1. Modify your cluster by creating new cache nodes in the Availability Zones where you want them. In your request, do the following:

   - Set `AZMode` (CLI: `--az-mode`) to `cross-az`.
   - Set `NumCacheNodes` (CLI: `--num-cache-nodes`) to the number of currently active cache nodes plus the number of new cache nodes you want to create.
   - Set `NewAvailabilityZones` (CLI: `--new-availability-zones`) to a list of the zones you want the new cache nodes created in. To let ElastiCache determine the Availability Zone for each new node, don't specify a list.
   - Set `ApplyImmediately` (CLI: `--apply-immediately`) to true.

   > **Note**
   >
   > If you are not using auto discovery, be sure to update your client application with the new cache node endpoints.

   Before moving on to the next step, be sure the Memcached nodes are fully created and available.

2. Modify your cluster by removing the nodes you no longer want in the original Availability Zone. In your request, do the following:

   - Set `NumCacheNodes` (CLI: `--num-cache-nodes`) to the number of active cache nodes you want after this modification is applied.
   - Set `CacheNodeIdsToRemove` (CLI: `--nodes-to-remove`) to a list of the cache nodes you want to remove from the cluster.

     The number of cache node IDs listed must equal the number of currently active nodes minus the value in `NumCacheNodes`.
   - (Optional) Set `ApplyImmediately` (CLI: `--apply-immediately`) to true.

If you don't set `ApplyImmediately` (CLI: `- -apply-immediately`) to true, the node deletions will take place at your next maintenance window.

## Locating your nodes

Amazon ElastiCache supports locating all of a cluster's nodes in a single or multiple Availability Zones (AZs). Further, if you elect to locate your nodes in multiple AZs (recommended), ElastiCache enables you to either choose the AZ for each node, or allow ElastiCache to choose them for you.

By locating the nodes in different AZs, you eliminate the chance that a failure, such as a power outage, in one AZ will cause your entire system to fail. Testing has demonstrated that there is no significant latency difference between locating all nodes in one AZ or spreading them across multiple AZs.

You can specify an AZ for each node when you create a cluster, or by adding nodes when you modify an existing cluster. When specifying an AZ for each node while creating a cluster, the AZ must be available in that subnet group. For more information, see the following:

- Creating a cluster for Memcached
- Creating a cluster for Valkey or Redis OSS
- Modifying an ElastiCache cluster
- Adding nodes to an ElastiCache cluster

## Supported Regions & endpoints

Amazon ElastiCache is available in multiple AWS Regions. This means that you can launch ElastiCache clusters in locations that meet your requirements. For example, you can launch in the AWS Region closest to your customers, or launch in a particular AWS Region to meet certain legal requirements.

Each Region is designed to be completely isolated from the other Regions. Within each Region are multiple Availability Zones (AZ). ElastiCache Serverless caches automatically replicate data across multiple availability zones (except `us-west-1`, where data is replicated in two availability zones) for high availability. When designing your own ElastiCache cluster, you can choose to launch your nodes in different AZs to achieve fault tolerance. For more information on Regions and Availability Zones, see Choosing regions and availability zones for ElastiCache at the top of this topic.

### Regions where ElastiCache is supported

| Region Name/Region | Endpoint | Protocol | |
|---|---|---|---|
| US East (Ohio) Region<br>us-east-2 | elasticache.us-east-2.amazonaws.com | HTTPS | |
| US East (N. Virginia) Region<br>us-east-1 | elasticache.us-east-1.amazonaws.com | HTTPS | |
| US West (N. California) Region<br>us-west-1 | elasticache.us-west-1.amazonaws.com | HTTPS | |
| US West (Oregon) Region<br>us-west-2 | elasticache.us-west-2.amazonaws.com | HTTPS | |
| Canada (Central) Region<br>ca-central-1 | elasticache.ca-central-1.amazonaws.com | HTTPS | |
| Canada (West) Region<br>ca-west-1 | elasticache.ca-west-1.amazonaws.com | HTTPS | |
| Asia Pacific (Jakarta)<br>ap-southeast-3 | elasticache.ap-southeast-3.amazonaws.com | HTTPS | |
| Asia Pacific (Mumbai) Region | elasticache.ap-south-1.amazonaws.com | HTTPS | |

| Region Name/Region | Endpoint | Protocol | |
|---|---|---|---|
| ap-south-1 | | | |
| Asia Pacific (Hyderabad) Region<br>ap-south-2 | elasticache.ap-south-2.amazonaws.com | HTTPS | |
| Asia Pacific (Tokyo) Region<br>ap-northeast-1 | elasticache.ap-northeast-1.amazonaws.com | HTTPS | |
| Asia Pacific (Seoul) Region<br>ap-northeast-2 | elasticache.ap-northeast-2.amazonaws.com | HTTPS | |
| Asia Pacific (Osaka) Region<br>ap-northeast-3 | elasticache.ap-northeast-3.amazonaws.com | HTTPS | |
| Asia Pacific (Singapore) Region<br>ap-southeast-1 | elasticache.ap-southeast-1.amazonaws.com | HTTPS | |
| Asia Pacific (Sydney) Region<br>ap-southeast-2 | elasticache.ap-southeast-2.amazonaws.com | HTTPS | |
| Europe (Frankfurt) Region<br>eu-central-1 | elasticache.eu-central-1.amazonaws.com | HTTPS | |

| Region Name/Region | Endpoint | Protocol | |
|---|---|---|---|
| Europe (Zurich) Region<br>eu-central-2 | elasticache.eu-central-2.amazonaws.com | HTTPS | |
| Europe (Stockholm) Region<br>eu-north-1 | elasticache.eu-north-1.amazonaws.com | HTTPS | |
| Middle East (Bahrain) Region<br>me-south-1 | elasticache.me-south-1.amazonaws.com | HTTPS | |
| Middle East (UAE) Region<br>me-central-1 | elasticache.me-central-1.amazonaws.com | HTTPS | |
| Europe (Ireland) Region<br>eu-west-1 | elasticache.eu-west-1.amazonaws.com | HTTPS | |
| Europe (London) Region<br>eu-west-2 | elasticache.eu-west-2.amazonaws.com | HTTPS | |
| EU (Paris) Region<br>eu-west-3 | elasticache.eu-west-3.amazonaws.com | HTTPS | |

| Region Name/Region | Endpoint | Protocol | |
|---|---|---|---|
| Europe (Milan) Region<br>eu-south-1 | elasticache.eu-south-1.amazonaws.com | HTTPS | |
| Europe (Spain) Region<br>eu-south-2 | elasticache.eu-south-2.amazonaws.com | HTTPS | |
| South America (São Paulo) Region<br>sa-east-1 | elasticache.sa-east-1.amazonaws.com | HTTPS | |
| China (Beijing) Region<br>cn-north-1 | elasticache.cn-north-1.amazonaws.com.cn | HTTPS | |
| China (Ningxia) Region<br>cn-northwest-1 | elasticache.cn-northwest-1.amazonaws.com.cn | HTTPS | |
| Asia Pacific (Hong Kong) Region<br>ap-east-1 | elasticache.ap-east-1.amazonaws.com | HTTPS | |
| Africa (Cape Town) Region<br>af-south-1 | elasticache.af-south-1.amazonaws.com | HTTPS | |

| Region Name/Region | Endpoint | Protocol | |
|---|---|---|---|
| Israel (Tel Aviv) Region<br><br>`il-central-1` | `elasticache.il-central-1.amazonaws.com` | HTTPS | |
| AWS GovCloud (US-West)<br><br>`us-gov-west-1` | `elasticache.us-gov-west-1.amazonaws.com` | HTTPS | |
| AWS GovCloud (US-East)<br><br>`us-gov-east-1` | `elasticache.us-gov-east-1.amazonaws.com` | HTTPS | |
| For information on using the AWS GovCloud (US) with ElastiCache, see Services in the AWS GovCloud (US) region: ElastiCache. | | | |

Some Regions support a subset of node types. For a table of supported node types by AWS Region, see Supported node types by AWS Region.

Most Regions support establishing a private connection between your VPC and ElastiCache API endpoints, by creating an interface VPC endpoint through AWS PrivateLink. For more information, see ElastiCache API and interface VPC endpoints (AWS PrivateLink).

For a table of AWS products and services by region, see Products and Services by Region.

## Using local zones with ElastiCache

A *Local Zone* is an extension of an AWS Region that is geographically close to your users. You can extend any virtual private cloud (VPC) from a parent AWS Region into a Local Zones by creating a new subnet and assigning it to the Local Zone. When you create a subnet in a Local Zone, your VPC is extended to that Local Zone. The subnet in the Local Zone operates the same as other subnets in your VPC.

By using Local Zones, you can place resources such as an ElastiCache cluster in multiple locations close to your users.

When you create an ElastiCache cluster, you can choose a subnet in a Local Zone. Local Zones have their own connections to the internet and support Direct Connect. Thus, resources created in a Local Zone can serve local users with very low-latency communications. For more information, see AWS Local Zones.

A Local Zone is represented by an AWS Region code followed by an identifier that indicates the location, for example `us-west-2-lax-1a`.

At this time, the available Local Zones are `us-west-2-lax-1a` and `us-west-2-lax-1b`.

The following limitations apply to ElastiCache for Local Zones:

- Global datastores aren't supported.
- Online migration isn't supported.
- The following node types are supported by Local Zones at this time:
  - Current generation:

    **M5 node types:** `cache.m5.large`, `cache.m5.xlarge`, `cache.m5.2xlarge`, `cache.m5.4xlarge`, `cache.m5.12xlarge`, `cache.m5.24xlarge`

    **R5 node types:** `cache.r5.large`, `cache.r5.xlarge`, `cache.r5.2xlarge`, `cache.r5.4xlarge`, `cache.r5.12xlarge`, `cache.r5.24xlarge`

    **T3 node types:** `cache.t3.micro`, `cache.t3.small`, `cache.t3.medium`

## Enabling a local zone

1. Enable the Local Zone in the Amazon EC2 console.

   For more information, see Enabling Local Zones in the *Amazon EC2 User Guide*.
2. Create a subnet in the Local Zone.

   For more information, see Creating a subnet in your VPC in the *Amazon VPC User Guide*.
3. Create an ElastiCache subnet group in the Local Zone.

   When you create an ElastiCache subnet group, choose the Availability Zone group for the Local Zone.

   For more information, see Creating a subnet group.

4. Create an ElastiCache for Memcached cluster that uses the ElastiCache subnet in the Local Zone.

   For more information, see Creating a Memcached cluster (console).

5. Create an ElastiCache for Redis OSS cluster that uses the ElastiCache subnet in the Local Zone. For more information, see one of the following topics:

   - Creating a Valkey (cluster mode disabled) cluster (Console)
   - Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console)

## Using Outposts with ElastiCache

You can use AWS Outposts with ElastiCache. Outposts is a fully managed service that extends AWS infrastructure, services, APIs, and tools to customer premises. By providing local access to AWS managed infrastructure, AWS Outposts enables customers to build and run applications on premises using the same programming interfaces as in AWS Regions, while using local compute and storage resources for lower latency and local data processing needs. An Outpost is a pool of AWS compute and storage capacity deployed at a customer site. AWS operates, monitors, and manages this capacity as part of an AWS Region. You can create subnets on your Outpost and specify them when you create AWS resources such as ElastiCache clusters.

**Note**

In this version, the following limitations apply:

- ElastiCache for Outposts only supports M5 and R5 node families.
- Multi-AZ (cross Outpost replication is not supported).
- Live migration is not supported.
- Local snapshots are not supported.
- Engine logs and slow logs can't be enabled.
- ElastiCache on Outposts does not support CoIP.
- ElastiCache for Outposts is not supported in the following regions: cn-north-1, cn-northwest-1 and ap-northeast-3.

## Using Outposts with the ElastiCache console

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose either **Valkey caches**, **Redis OSS caches**, or **Memcached caches**.
3. If you chose **Valkey caches**, select **Create Valkey cache**. If you chose **Redis OSS caches**, select **Create Redis OSS cache**. If you chose **Memcached caches**, select **Create Memcached cache**.
4. Under **Cluster settings**, select **Design your own cache** and **Cluster cache**. Leave **Cluster mode** set as **Disabled**. Then create a name and optional description for the cache.
5. For location, choose **On premises**.
6. In On-premises section you will see the field **Outpost ID**. Enter the ID for where the cluster will run.

   All further settings under **Cluster settings** can stay as default.
7. In **Connectivity**, select **Create a new subnet group** and enter the **VPC ID**. Leave the rest as default, and select **Next**.

### Configure on-premises options

You can select either an available Outpost to add your cluster or, if there are no available Outposts, create a new one using the following steps:

**Under On-Premises options:**

1. Under **Valkey settings**, **Redis OSS settings**, or **Memcached settings**, depending on your preferred engine:
   a. **Name**: Enter a name for the cluster
   b. **Description**: Enter a description for the cluster.
   c. **Engine version compatilbility**: Engine version is based on the AWS Outpost region
   d. **Port**: For Valkey or Redis OSS, accept the default port 6379. For Memcached, accept the default port 11211. If you prefer to use a different port, type the port number.
   e. **Parameter group**: Use the drop-down to select a default or custom parameter group.
   f. **Node Type**: Available instances are based on Outposts availability. If using Valkey or Redis OSS, Porting Assistant for .NET for Outposts only supports M5 and R5 node families. From

the drop down list, select **Outposts** and then select an available node type you want to use for this cluster. Then select **Save**.

g. **Number of Replicas**: Enter the number of read replicas you want created for this replication group. You must have at least one and no more than five read replicas. The default value is 2.

The autogenerated names of the read replicas follow the same pattern as that of the primary cluster's name, with a dash and sequential three-digit number added to the end, beginning with `-002`. For example, if your replication group is named `MyGroup`, then the names of the secondaries would be `MyGroup-002`, `MyGroup-003`, `MyGroup-004`, `MyGroup-005`, `MyGroup-006`.

2. Under **Connectivity**:

a. **Subnet Group**: From the list, select **Create new**.

- **Name**: Enter a name for the subnet group
- **Description**: Enter a description for the subnet group
- **VPC ID**: The VPC ID should match the Outpost VPC. If you select a VPC that has no subnet IDs on the Outposts, the list will return empty.
- **Availability Zone or Outpost**: Select the Outpost you are using.
- **Subnet ID**: Select a subnet ID that is available for the Outpost. If there are no subnet IDs available, you need to create them. For more information, see Create a Subnet.

b. Select **Create**.

### Viewing Outpost cluster details

On the list page, select a cluster that belongs to an AWS Outpost and note the following when viewing the **Cluster details**:

- **Availability Zone**: This will represent the Outpost, using an ARN (Amazon Resource Name) and the AWS Resource Number.
- **Outpost name**: The name of the AWS Outpost.

## Using Outposts with the AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to control multiple AWS services from the command line and automate them through scripts. You can use the AWS CLI for ad hoc (one-time) operations.

### Downloading and configuring the AWS CLI

The AWS CLI runs on Windows, macOS, or Linux. Use the following procedure to download and configure it.

### To download, install, and configure the CLI

1. Download the AWS CLI on the AWS Command Line Interface webpage.
2. Follow the instructions for Installing the AWS CLI and Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

### Using the AWS CLI with Outposts

Use the following CLI operation to create a cluster that uses Outposts:

- create-cache-cluster – Using this operation, the `outpost-mode` parameter accepts a value that specifies whether the nodes in the cluster are created in a single Outpost or across multiple Outposts.

> **Note**
>
> At this time, only `single-outpost` mode is supported.

```
aws elasticache create-cache-cluster \
   --cache-cluster-id cluster id \
   --outpost-mode single-outpost \
```

# Working with ElastiCache

In this section you can find details about how to manage the various components of your ElastiCache implementation.

**Topics**


- Snapshot and restore
- Engine versions and upgrading in ElastiCache
- ElastiCache best practices and caching strategies
- Durability in ElastiCache
- Managing your node-based cluster in ElastiCache
- Connecting an EC2 instance and an ElastiCache cache automatically
- Scaling ElastiCache
- Getting started with Bloom filters
- Getting started with Watch in Serverless
- Getting started with Search
- Getting started with data aggregations
- Getting started with JSON for Valkey and Redis OSS
- Tagging your ElastiCache resources
- Using the Amazon ElastiCache Well-Architected Lens
- Common troubleshooting steps and best practices with ElastiCache


# Snapshot and restore

Amazon ElastiCache caches running Valkey, Redis OSS, or Serverless Memcached can back up their data by creating a snapshot. You can use the backup to restore a cache or seed data to a new cache. The backup consists of the cache's metadata, along with all of the data in the cache. All backups are written to Amazon Simple Storage Service (Amazon S3), which provides durable storage. At any time, you can restore your data by creating a new Valkey, Redis OSS, or Serverless Memcached cache and populating it with data from a backup. With ElastiCache, you can manage backups using the AWS Management Console, the AWS Command Line Interface (AWS CLI), and the ElastiCache API.

If you plan to delete a cache and it's important to preserve the data, you can take an extra precaution. To do this, create a manual backup first, verify that its status is *available*, and then delete the cache. Doing this makes sure that if the backup fails, you still have the cache data available. You can retry making a backup, following the best practices outlined preceding.

**Topics**


- Backup constraints
- Performance impact of backups of node-based clusters
- Scheduling automatic backups
- Taking manual backups
- Creating a final backup
- Describing backups
- Copying backups
- Exporting a backup
- Restoring from a backup into a new cache
- Deleting a backup
- Tagging backups
- Tutorial: Seeding a new node-based cluster with an externally created backup


## Backup constraints

Consider the following constraints when planning or making backups:

- Backup and restore are supported only for caches running on Valkey, Redis OSS or Serverless Memcached.
- For Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled) clusters, backup and restore are supported for all node types.
- During any contiguous 24-hour period, you can create no more than 24 manual backups per serverless cache. For Valkey and Redis OSS node-based clusters, you can create no more than 20 manual backups per node in the cluster.
- Valkey or Redis OSS (cluster mode enabled) only supports taking backups on the cluster level (for the API or CLI, the replication group level). Valkey or Redis OSS (cluster mode enabled) doesn't support taking backups at the shard level (for the API or CLI, the node group level).

- During the backup process, you can't run any other API or CLI operations on serverless cache. You can run API or CLI operations on a node-based cluster during backup.
- If you are using Valkey or Redis OSS caches with data tiering, you cannot export a backup to Amazon S3.
- You can restore a backup from a data tiering cluster to another cluster that also uses data tiering.

## Performance impact of backups of node-based clusters

Backups on serverless caches are transparent to the application with no performance impact. However, when creating backups for node-based clusters, there can be some performance impact depending on the available reserved memory. Backups for node-based clusters are not available with ElastiCache for Memcached but are available with ElastiCache for Redis OSS.

The following are guidelines for improving backup performance for node-based clusters.

- Set the `reserved-memory-percent` parameter – To mitigate excessive paging, we recommend that you set the *reserved-memory-percent* parameter. This parameter prevents Valkey and Redis OSS from consuming all of the node's available memory, and can help reduce the amount of paging. You might also see performance improvements by simply using a larger node. For more information about the *reserved-memory* and *reserved-memory-percent* parameters, see Managing reserved memory for Valkey and Redis OSS.

- Create backups from a read replica – If you are running Valkey or Redis OSS in a node group with more than one node, you can take a backup from the primary node or one of the read replicas. Because of the system resources required during BGSAVE, we recommend that you create backups from one of the read replicas. While the backup is being created from the replica, the primary node remains unaffected by BGSAVE resource requirements. The primary node can continue serving requests without slowing down.

  To do this, see Creating a manual backup (Console) and in the **Cluster Name** field in the **Create Backup** window, choose a replica instead of the default primary node.

If you delete a replication group and request a final backup, ElastiCache always takes the backup from the primary node. This ensures that you capture the very latest Valkey or Redis OSS data, before the replication group is deleted.

# Scheduling automatic backups

You can enable automatic backups for any Valkey or Redis OSS serverless cache or node-based cluster. When automatic backups are enabled, ElastiCache creates a backup of the cache on a daily basis. There is no impact on the cache and the change is immediate. Automatic backups can help guard against data loss. In the event of a failure, you can create a new cache, restoring your data from the most recent backup. The result is a warm-started cache, preloaded with your data and ready for use. For more information, see Restoring from a backup into a new cache.

You can enable automatic backups for any Memcached Serverless cache. When automatic backups are enabled, ElastiCache creates a backup of the cache on a daily basis. There is no impact on the cache and the change is immediate. Automatic backups can help guard against data loss. In the event of a failure, you can create a new cache, restoring your data from the most recent backup. The result is a warm-started cache, preloaded with your data and ready for use. For more information, see Restoring from a backup into a new cache.

When you schedule automatic backups, you should plan the following settings:

- **Backup start time** – A time of day when ElastiCache begins creating a backup. You can set the backup window for any time when it's most convenient. If you don't specify a backup window, ElastiCache assigns one automatically.

- **Backup retention limit** – The number of days the backup is retained in Amazon S3. For example, if you set the retention limit to 5, then a backup taken today is retained for 5 days. When the retention limit expires, the backup is automatically deleted.

  The maximum backup retention limit is 35 days. If the backup retention limit is set to 0, automatic backups are disabled for the cache.

When you schedule automatic backups, ElastiCache will begin creating the backup. You can set the backup window for any time when it's most convenient. If you don't specify a backup window, ElastiCache assigns one automatically.

You can enable or disable automatic backups when either creating a new cache or updating an existing cache, by using the ElastiCache console, the AWS CLI, or the ElastiCache API. For Valkey and Redis OSS, this is done by checking the **Enable Automatic Backups** box in the **Advanced**

**Valkey Settings** or **Advanced Redis OSS Settings** section. For Memcached, this is done by checking the **Enable Automatic Backups** box in the **Advanced Memcached Settings** section.

# Taking manual backups

In addition to automatic backups, you can create a *manual* backup at any time. Unlike automatic backups, which are automatically deleted after a specified retention period, manual backups do not have a retention period after which they are automatically deleted. Even if you delete the cache, any manual backups from that cache are retained. If you no longer want to keep a manual backup, you must explicitly delete it yourself.

In addition to directly creating a manual backup, you can create a manual backup in one of the following ways:

- Copying backups. It does not matter whether the source backup was created automatically or manually.
- Creating a final backup. Create a backup immediately before deleting a cluster or node.

You can create a manual backup of a cache using the AWS Management Console, the AWS CLI, or the ElastiCache API.

You can generate manual backups from replicas that are cluster mode enabled, and cluster mode disabled.

### Creating a manual backup (Console)

**To create a backup of a cache (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey caches**, **Redis OSS caches**, or **Memcached caches**, depending on your preference.
3. Choose the box to the left of the name of the cache you want to back up.
4. Choose **Backup**.
5. In the **Create Backup** dialog, type in a name for your backup in the **Backup Name** box. We recommend that the name indicate which cluster was backed up and the date and time the backup was made.

   Cluster naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

6. Choose **Create Backup**.

   The status of the cluster changes to *snapshotting*.

### Creating a manual backup (AWS CLI)

#### Manual backup of a serverless cache with the AWS CLI

To create a manual backup of a cache using the AWS CLI, use the `create-serverless-cache-snapshot` AWS CLI operation with the following parameters:

- `--serverless-cache-name` – The name of the serverless cache that you are backing up.
- `--serverless-cache-snapshot-name` – Name of the snapshot to be created.

For Linux, macOS, or Unix:

```
aws elasticache create-serverless-cache-snapshot \
                        --serverless-cache-name CacheName \
                        --serverless-cache-snapshot-name bkup-20231127
```

For Windows:

```
aws elasticache create-serverless-cache-snapshot ^
    --serverless-cache-name CacheName ^
    --serverless-cache-snapshot-name bkup-20231127
```

#### Manual backup of a node-based cluster with the AWS CLI

To create a manual backup of a node-based cluster using the AWS CLI, use the `create-snapshot` AWS CLI operation with the following parameters:

- `--cache-cluster-id`

- If the cluster you're backing up has no replica nodes, `--cache-cluster-id` is the name of the cluster you are backing up, for example *`mycluster`*.
- If the cluster you're backing up has one or more replica nodes, `--cache-cluster-id` is the name of the node in the cluster that you want to use for the backup. For example, the name might be *`mycluster-002`*.

  Use this parameter only when backing up a Valkey or Redis OSS (cluster mode disabled) cluster.

- `--replication-group-id` – Name of the Valkey or Redis OSS (cluster mode enabled) cluster (CLI/API: a replication group) to use as the source for the backup. Use this parameter when backing up a Valkey or Redis OSS (cluster mode enabled) cluster.

- `--snapshot-name` – Name of the snapshot to be created.

  Cluster naming constraints are as follows:

  - Must contain 1–40 alphanumeric characters or hyphens.
  - Must begin with a letter.
  - Can't contain two consecutive hyphens.
  - Can't end with a hyphen.

**Example 1: Backing up a Valkey or Redis OSS (Cluster Mode Disabled) cluster that has no replica nodes**

The following AWS CLI operation creates the backup `bkup-20150515` from the Valkey or Redis OSS (cluster mode disabled) cluster `myNonClusteredRedis` that has no read replicas.

For Linux, macOS, or Unix:

```
aws elasticache create-snapshot \
    --cache-cluster-id myNonClusteredRedis \
    --snapshot-name bkup-20150515
```

For Windows:

```
aws elasticache create-snapshot ^
```

```
    --cache-cluster-id myNonClusteredRedis ^
    --snapshot-name bkup-20150515
```

**Example 2: Backing up a Valkey or Redis OSS (Cluster Mode Disabled) cluster with replica nodes**

The following AWS CLI operation creates the backup `bkup-20150515` from the Valkey or Redis OSS (cluster mode disabled) cluster `myNonClusteredRedis`. This backup has one or more read replicas.

For Linux, macOS, or Unix:

```
aws elasticache create-snapshot \
    --cache-cluster-id myNonClusteredRedis-001 \
    --snapshot-name bkup-20150515
```

For Windows:

```
aws elasticache create-snapshot ^
    --cache-cluster-id myNonClusteredRedis-001 ^
    --snapshot-name bkup-20150515
```

**Example Output: Backing Up a Valkey or Redis OSS (Cluster Mode Disabled) Cluster with Replica Nodes**

Output from the operation looks something like the following.

```
{
    "Snapshot": {
        "Engine": "redis",
        "CacheParameterGroupName": "default.redis6.x",
        "VpcId": "vpc-91280df6",
        "CacheClusterId": "myNonClusteredRedis-001",
        "SnapshotRetentionLimit": 0,
        "NumCacheNodes": 1,
        "SnapshotName": "bkup-20150515",
        "CacheClusterCreateTime": "2017-01-12T18:59:48.048Z",
        "AutoMinorVersionUpgrade": true,
        "PreferredAvailabilityZone": "us-east-1c",
        "SnapshotStatus": "creating",
        "SnapshotSource": "manual",
        "SnapshotWindow": "08:30-09:30",
        "EngineVersion": "6.0",
```

```
        "NodeSnapshots": [
            {
                "CacheSize": "",
                "CacheNodeId": "0001",
                "CacheNodeCreateTime": "2017-01-12T18:59:48.048Z"
            }
        ],
        "CacheSubnetGroupName": "default",
        "Port": 6379,
        "PreferredMaintenanceWindow": "wed:07:30-wed:08:30",
        "CacheNodeType": "cache.m5.2xlarge",
        "DataTiering": "disabled"
    }
}
```

**Example 3: Backing up a cluster for Valkey or Redis OSS (Cluster Mode Enabled)**

The following AWS CLI operation creates the backup `bkup-20150515` from the Valkey or Redis OSS (cluster mode enabled) cluster `myClusteredRedis`. Note the use of `--replication-group-id` instead of `--cache-cluster-id` to identify the source. Also note that ElastiCache takes the backup using the replica node when present, and will default to the primary node if a replica node is unavailable.

For Linux, macOS, or Unix:

```
aws elasticache create-snapshot \
    --replication-group-id myClusteredRedis \
    --snapshot-name bkup-20150515
```

For Windows:

```
aws elasticache create-snapshot ^
    --replication-group-id myClusteredRedis ^
    --snapshot-name bkup-20150515
```

**Example Output: Backing Up a Valkey or Redis OSS (Cluster Mode Enabled) Cluster**

Output from this operation looks something like the following.

```
{
    "Snapshot": {
```

```
        "Engine": "redis",
        "CacheParameterGroupName": "default.redis6.x.cluster.on",
        "VpcId": "vpc-91280df6",
        "NodeSnapshots": [
            {
                "CacheSize": "",
                "NodeGroupId": "0001"
            },
            {
                "CacheSize": "",
                "NodeGroupId": "0002"
            }
        ],
        "NumNodeGroups": 2,
        "SnapshotName": "bkup-20150515",
        "ReplicationGroupId": "myClusteredRedis",
        "AutoMinorVersionUpgrade": true,
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "enabled",
        "SnapshotStatus": "creating",
        "SnapshotSource": "manual",
        "SnapshotWindow": "10:00-11:00",
        "EngineVersion": "6.0",
        "CacheSubnetGroupName": "default",
        "ReplicationGroupDescription": "2 shards 2 nodes each",
        "Port": 6379,
        "PreferredMaintenanceWindow": "sat:03:30-sat:04:30",
        "CacheNodeType": "cache.r5.large",
        "DataTiering": "disabled"
    }
}
```

### Related topics

For more information, see create-snapshot in the *AWS CLI Command Reference*.

### Creating a backup using CloudFormation

You can use CloudFormation to create a backup of your ElastiCache Redis OSS or Valkey cache, using the `AWS::ElastiCache::ServerlessCache` or `AWS::ElastiCache::ReplicationGroup` properties.

### Using the `AWS::ElastiCache::ServerlessCache` resource

Use this to create a backup using the AWS::ElastiCache::ServerlessCache resource:

```
Resources:
                    iotCatalog:
                        Type: AWS::ElastiCache::ServerlessCache
                            Properties:
                            ...
                            ServerlessCacheName: "your-cache-name"
                            Engine: "redis"
                            CacheUsageLimits
```

### Using the AWS::ElastiCache::ReplicationGroup resource

Use the `AWS::ElastiCache::ReplicationGroup` resource:

```
Resources:
                    iotCatalog:
                        Type: AWS::ElastiCache::ReplicationGroup
                            Properties:
                            ...
                            ReplicationGroupDescription: "Description of your
 replication group"
                            Engine: "redis"
                            CacheNodeType
                            NumCacheClusters
                            AutomaticFailoverEnabled
                            AtRestEncryptionEnabled
```

# Creating a final backup

You can create a final backup using the ElastiCache console, the AWS CLI, or the ElastiCache API.

### Creating a final backup (Console)

You can create a final backup when you delete a Valkey, Memcached, or Redis OSS serverless cache, or a Valkey or Redis OSS node-based cluster, by using the ElastiCache console.

To create a final backup when deleting a cache, on the delete dialog box choose **Yes** under **Create backup** and give the backup a name.

### Related topics

- Using the AWS Management Console
- Deleting a Replication Group (Console)

### Creating a final backup (AWS CLI)

You can create a final backup when deleting a cache using the AWS CLI.

### Topics

- When deleting a Valkey cache, Memcached serverless cache, or Redis OSS cache
- When deleting a Valkey or Redis OSS cluster with no read replicas
- When deleting a Valkey or Redis OSS cluster with read replicas

### When deleting a Valkey cache, Memcached serverless cache, or Redis OSS cache

To create a final backup, use the `delete-serverless-cache` AWS CLI operation with the following parameters.

- `--serverless-cache-name` – Name of the cache being deleted.
- `--final-snapshot-name` – Name of the backup.

The following code creates the final backup `bkup-20231127-final` when deleting the cache `myserverlesscache`.

For Linux, macOS, or Unix:

```
aws elasticache delete-serverless-cache \
        --serverless-cache-name myserverlesscache \
        --final-snapshot-name bkup-20231127-final
```

For Windows:

```
aws elasticache delete-serverless-cache ^
        --serverless-cache-name myserverlesscache ^
        --final-snapshot-name bkup-20231127-final
```

For more information, see delete-serverless-cache in the *AWS CLI Command Reference*.

**When deleting a Valkey or Redis OSS cluster with no read replicas**

To create a final backup for a node-based cluster with no read replicas, use the `delete-cache-cluster` AWS CLI operation with the following parameters.

- `--cache-cluster-id` – Name of the cluster being deleted.
- `--final-snapshot-identifier` – Name of the backup.

The following code creates the final backup `bkup-20150515-final` when deleting the cluster `myRedisCluster`.

For Linux, macOS, or Unix:

```
aws elasticache delete-cache-cluster \
        --cache-cluster-id myRedisCluster \
        --final-snapshot-identifier bkup-20150515-final
```

For Windows:

```
aws elasticache delete-cache-cluster ^
        --cache-cluster-id myRedisCluster ^
        --final-snapshot-identifier bkup-20150515-final
```

For more information, see delete-cache-cluster in the *AWS CLI Command Reference*.

**When deleting a Valkey or Redis OSS cluster with read replicas**

To create a final backup when deleting a replication group, use the `delete-replication-group` AWS CLI operation, with the following parameters:

- `--replication-group-id` – Name of the replication group being deleted.
- `--final-snapshot-identifier` – Name of the final backup.

The following code takes the final backup `bkup-20150515-final` when deleting the replication group `myReplGroup`.

For Linux, macOS, or Unix:

```
aws elasticache delete-replication-group \
        --replication-group-id myReplGroup \
        --final-snapshot-identifier bkup-20150515-final
```

For Windows:

```
aws elasticache delete-replication-group ^
        --replication-group-id myReplGroup ^
        --final-snapshot-identifier bkup-20150515-final
```

For more information, see delete-replication-group in the *AWS CLI Command Reference*.

# Describing backups

The following procedures show you how to display a list of your backups. If you desire, you can also view the details of a particular backup.

### Describing backups (Console)

#### To display backups using the AWS Management Console

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Backups**.
3. To see the details of a particular backup, choose the box to the left of the backup's name.

### Describing serverless backups (AWS CLI)

To display a list of serverless backups and optionally details about a specific backup, use the `describe-serverless-cache-snapshots` CLI operation.

#### Examples

The following operation uses the parameter `--max-records` to list up to 20 backups associated with your account. Omitting the parameter `--max-records` lists up to 50 backups.

```
aws elasticache describe-serverless-cache-snapshots --max-records 20
```

The following operation uses the parameter `--serverless-cache-name` to list only the backups associated with the cache `my-cache`.

```
aws elasticache describe-serverless-cache-snapshots --serverless-cache-name my-cache
```

The following operation uses the parameter `--serverless-cache-snapshot-name` to display the details of the backup `my-backup`.

```
aws elasticache describe-serverless-cache-snapshots --serverless-cache-snapshot-
name my-backup
```

For more information, see describe-serverless-cache-snapshots in the AWS CLI Command Reference.

### Describing node-based cluster backups (AWS CLI)

To display a list of node-based cluster backups and optionally details about a specific backup, use the `describe-snapshots` CLI operation.

#### Examples

The following operation uses the parameter `--max-records` to list up to 20 backups associated with your account. Omitting the parameter `--max-records` lists up to 50 backups.

```
aws elasticache describe-snapshots --max-records 20
```

The following operation uses the parameter `--cache-cluster-id` to list only the backups associated with the cluster `my-cluster`.

```
aws elasticache describe-snapshots --cache-cluster-id my-cluster
```

The following operation uses the parameter `--snapshot-name` to display the details of the backup `my-backup`.

```
aws elasticache describe-snapshots --snapshot-name my-backup
```

For more information, see describe-snapshots in the AWS CLI Command Reference.

# Copying backups

You can create a copy of any backup, whether it was created automatically or manually. You can also export your backup so you can access it from outside ElastiCache. For guidance on exporting your backup, see Exporting a backup.

The following steps show you how to copy a backup.

### Copying backups (Console)

### To copy a backup (console)

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of your backups, from the left navigation pane choose **Backups**.
3. From the list of backups, choose the box to the left of the name of the backup you want to copy.
4. Choose **Actions** then **Copy**.
5. In the **New backup name** box, type a name for your new backup.
6. Choose **Copy**.

### Copying a serverless backup (AWS CLI)

To copy a backup of a serverless cache, use the `copy-serverless-cache-snapshot` operation.

### Parameters

- `--source-serverless-cache-snapshot-name` – Name of the backup to be copied.
- `--target-serverless-cache-snapshot-name` – Name of the backup's copy.

The following example makes a copy of an automatic backup.

For Linux, macOS, or Unix:

```
aws elasticache copy-serverless-cache-snapshot \
    --source-serverless-cache-snapshot-name automatic.my-cache-2023-11-27-03-15 \
    --target-serverless-cache-snapshot-name my-backup-copy
```

For Windows:

```
aws elasticache copy-serverless-cache-snapshot ^
    --source-serverless-cache-snapshot-name automatic.my-cache-2023-11-27-03-15 ^
    --target-serverless-cache-snapshot-name my-backup-copy
```

For more information, see copy-serverless-cache-snapshot in the *AWS CLI*.

**Copying a node-based cluster backup (AWS CLI)**

To copy a backup of a node-based cluster, use the `copy-snapshot` operation.

**Parameters**

- `--source-snapshot-name` – Name of the backup to be copied.
- `--target-snapshot-name` – Name of the backup's copy.
- `--target-bucket` – Reserved for exporting a backup. Do not use this parameter when making a copy of a backup. For more information, see Exporting a backup.

The following example makes a copy of an automatic backup.

For Linux, macOS, or Unix:

```
aws elasticache copy-snapshot  \
    --source-snapshot-name automatic.my-redis-primary-2014-03-27-03-15 \
    --target-snapshot-name amzn-s3-demo-bucket
```

For Windows:

```
aws elasticache copy-snapshot ^
    --source-snapshot-name automatic.my-redis-primary-2014-03-27-03-15 ^
    --target-snapshot-name amzn-s3-demo-bucket
```

For more information, see copy-snapshot in the *AWS CLI*.

# Exporting a backup

Amazon ElastiCache supports exporting your ElastiCache for Redis OSS backup to an Amazon Simple Storage Service (Amazon S3) bucket, which gives you access to it from outside ElastiCache. You can export a backup using the ElastiCache console, the AWS CLI, or the ElastiCache API.

Exporting a backup can be helpful if you need to launch a cluster in another AWS Region. You can export your data in one AWS Region, copy the .rdb file to the new AWS Region, and then use that .rdb file to seed the new cache instead of waiting for the new cluster to populate through use. For information about seeding a new cluster, see Tutorial: Seeding a new node-based cluster with an externally created backup. Another reason you might want to export your cache's data is to use the .rdb file for offline processing.

**Important**

- The ElastiCache backup and the Amazon S3 bucket that you want to copy it to must be in the same AWS Region.

  Though backups copied to an Amazon S3 bucket are encrypted, we strongly recommend that you do not grant others access to the Amazon S3 bucket where you want to store your backups.
- Exporting a backup to Amazon S3 is not supported for clusters using data tiering. For more information, see Data tiering in ElastiCache.
- Exporting a backup is available for: node-based Valkey clusters, node-based Redis OSS clusters, and Valkey, Memcached, and Redis OSS serverless caches. Exporting a backup is not available for node-based Memcached clusters.

Before you can export a backup to an Amazon S3 bucket, you must have an Amazon S3 bucket in the same AWS Region as the backup. Grant ElastiCache access to the bucket. The first two steps show you how to do this.

## Create an Amazon S3 bucket

The following steps use the Amazon S3 console to create an Amazon S3 bucket where you export and store your ElastiCache backup.

**To create an Amazon S3 bucket**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create Bucket**.
3. In **Create a Bucket - Select a Bucket Name and Region**, do the following:

   a. In **Bucket Name**, type a name for your Amazon S3 bucket.

      The name of your Amazon S3 bucket must be DNS-compliant. Otherwise, ElastiCache can't access your backup file. The rules for DNS compliance are:

      - Names must be at least 3 and no more than 63 characters long.
      - Names must be a series of one or more labels separated by a period (.) where each label:
        - Starts with a lowercase letter or a number.
        - Ends with a lowercase letter or a number.
        - Contains only lowercase letters, numbers, and dashes.
      - Names can't be formatted as an IP address (for example, 192.0.2.0).

   b. From the **Region** list, choose an AWS Region for your Amazon S3 bucket. This AWS Region must be the same AWS Region as the ElastiCache backup you want to export.

   c. Choose **Create**.

For more information about creating an Amazon S3 bucket, see Creating a bucket in the *Amazon Simple Storage Service User Guide*.

## Grant ElastiCache access to your Amazon S3 bucket

ElastiCache requires access to your Amazon S3 bucket to copy a snapshot to it. We recommend granting access by using an Amazon S3 bucket policy rather than access control lists (ACLs).

> **Warning**
>
> Even though backups copied to an Amazon S3 bucket are encrypted, your data can be accessed by anyone with access to your Amazon S3 bucket. Therefore, we strongly recommend that you set up IAM policies to prevent unauthorized access to this Amazon S3 bucket. For more information, see Managing access in the *Amazon S3 User Guide*.

Add the following bucket policy to your Amazon S3 bucket. Replace *amzn-s3-demo-bucket* with the name of your Amazon S3 bucket and *region* with the AWS Region of your bucket (for example, `us-east-1`).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ElastiCacheSnapshotExport",
            "Effect": "Allow",
            "Principal": {
                "Service": "region.elasticache-snapshot.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:ListMultipartUploadParts",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

**To add the bucket policy using the Amazon S3 console**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the name of the Amazon S3 bucket that you want to copy the backup to. This should be the S3 bucket that you created in Create an Amazon S3 bucket.
3. Choose the **Permissions** tab.
4. Under **Bucket policy**, choose **Edit**.

5. Paste the bucket policy into the policy editor. Replace the *region* and *amzn-s3-demo-bucket* placeholders with your values.
6. Choose **Save changes**.

For more information about migrating from ACLs to bucket policies, see Grant Amazon ElastiCache (Redis OSS) access to your S3 bucket in the *Amazon S3 User Guide*.

### Required IAM permissions for the caller

To call the export operation, you must have permissions to call `elasticache:CopySnapshot` and to access the target Amazon S3 bucket. You can call this operation using the ElastiCache console, the AWS CLI, or the ElastiCache API `CopySnapshot` operation. These IAM permissions are separate from the Amazon S3 bucket policy that grants the ElastiCache service access to the bucket. For information about the bucket policy, see Grant ElastiCache access to your Amazon S3 bucket.

The following IAM policy shows the minimum permissions that you need. Replace *amzn-s3-demo-bucket* with the name of your target Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ElastiCacheCopySnapshot",
            "Effect": "Allow",
            "Action": "elasticache:CopySnapshot",
            "Resource": "*"
        },
        {
            "Sid": "S3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
```

```
        },
        {
            "Sid": "S3GlobalAccess",
            "Effect": "Allow",
            "Action": "s3:ListAllMyBuckets",
            "Resource": "arn:aws:s3:::*"
        }
    ]
}
```

> **Important**
>
> The `s3:ListAllMyBuckets` action is an account-level operation. Its `Resource` element must be `"*"` or `"arn:aws:s3:::*"`. Scoping this action to a specific bucket ARN makes it ineffective and causes the error: "Elasticache was unable to validate the authenticated user has access on the S3 bucket." For more information, see Amazon S3 actions related to accounts in the *Amazon S3 User Guide*.

## Export an ElastiCache backup

Now you've created your S3 bucket and granted ElastiCache permissions to access it. Next, you can use the ElastiCache console, the AWS CLI, or the ElastiCache API to export your snapshot to it.

### Exporting an ElastiCache backup (Console)

The following steps use the ElastiCache console to export a backup to an Amazon S3 bucket so that you can access it from outside ElastiCache. The Amazon S3 bucket must be in the same AWS Region as the ElastiCache backup.

### To export an ElastiCache backup to an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of your backups, from the left navigation pane choose **Backups**.
3. From the list of backups, choose the box to the left of the name of the backup you want to export.
4. Choose **Export**.
5. In **Export Backup**, do the following:

a. In **New backup name** box, type a name for your new backup.

   The name must be between 1 and 1,000 characters and able to be UTF-8 encoded.

   ElastiCache adds an instance identifier and `.rdb` to the value that you enter here. For example, if you enter `my-exported-backup`, ElastiCache creates `my-exported-backup-0001.rdb`.

b. From the **Target S3 Location** list, choose the name of the Amazon S3 bucket that you want to copy your backup to (the bucket that you created in Create an Amazon S3 bucket).

   The **Target S3 Location** must be an Amazon S3 bucket in the backup's AWS Region with the following permissions for the export process to succeed.

   - Object access – **Read** and **Write**.
   - Permissions access – **Read**.

   For more information, see Grant ElastiCache access to your Amazon S3 bucket.

c. Choose **Copy**.

**Note**

If your S3 bucket does not have the permissions needed for ElastiCache to export a backup to it, you receive one of the following error messages. Return to Grant ElastiCache access to your Amazon S3 bucket to add the permissions specified and retry exporting your backup.

- ElastiCache has not been granted READ permissions %s on the S3 Bucket.

  **Solution:** Add Read permissions on the bucket.

- ElastiCache has not been granted WRITE permissions %s on the S3 Bucket.

  **Solution:** Add Write permissions on the bucket.

- ElastiCache has not been granted READ_ACP permissions %s on the S3 Bucket.

  **Solution:** Add **Read** for Permissions access on the bucket.

If you want to copy your backup to another AWS Region, use Amazon S3 to copy it. For more information, see Copying an object in the *Amazon Simple Storage Service User Guide*.

**Exporting an ElastiCache serverless backup (AWS CLI)**

**Exporting a backup of a serverless cache**

Export the backup to an Amazon S3 bucket using the `export-serverless-cache-snapshot` CLI operation with the following parameters:

**Parameters**

- `--serverless-cache-snapshot-name` – Name of the backup to be copied.
- `--s3-bucket-name` – Name of the Amazon S3 bucket where you want to export the backup. A copy of the backup is made in the specified bucket.

  The `--s3-bucket-name` must be an Amazon S3 bucket in the backup's AWS Region with the following permissions for the export process to succeed.
  - Object access – **Read** and **Write**.
  - Permissions access – **Read**.

The following operation copies a backup to my-s3-bucket.

For Linux, macOS, or Unix:

```
aws elasticache export-serverless-cache-snapshot \
    --serverless-cache-snapshot-name automatic.my-redis-2023-11-27 \
    --s3-bucket-name my-s3-bucket
```

For Windows:

```
aws elasticache export-serverless-cache-snapshot ^
    --serverless-cache-snapshot-name automatic.my-redis-2023-11-27 ^
    --s3-bucket-name my-s3-bucket
```

**Exporting a node-based ElastiCache cluster backup (AWS CLI)**

**Exporting a backup of a node-based cluster**

Export the backup to an Amazon S3 bucket using the `copy-snapshot` CLI operation with the following parameters:

**Parameters**

- `--source-snapshot-name` – Name of the backup to be copied.
- `--target-snapshot-name` – Name of the backup's copy.

  The name must be between 1 and 1,000 characters and able to be UTF-8 encoded.

  ElastiCache adds an instance identifier and `.rdb` to the value you enter here. For example, if you enter `my-exported-backup`, ElastiCache creates `my-exported-backup-0001.rdb`.
- `--target-bucket` – Name of the Amazon S3 bucket where you want to export the backup. A copy of the backup is made in the specified bucket.

  The `--target-bucket` must be an Amazon S3 bucket in the backup's AWS Region with the following permissions for the export process to succeed.
  - Object access – **Read** and **Write**.
  - Permissions access – **Read**.

  For more information, see Grant ElastiCache access to your Amazon S3 bucket.

The following operation copies a backup to my-s3-bucket.

For Linux, macOS, or Unix:

```
aws elasticache copy-snapshot \
    --source-snapshot-name automatic.my-redis-primary-2016-06-27-03-15 \
    --target-snapshot-name my-exported-backup \
    --target-bucket my-s3-bucket
```

For Windows:

```
aws elasticache copy-snapshot ^
    --source-snapshot-name automatic.my-redis-primary-2016-06-27-03-15 ^
    --target-snapshot-name my-exported-backup ^
    --target-bucket my-s3-bucket
```

# Restoring from a backup into a new cache

You can restore an existing backup from Valkey into a new Valkey cache or node-based cluster, and restore an existing Redis OSS backup into a new Redis OSS cache or node-based cluster. You can also restore an existing Memcached serverless cache backup into a new Memcached serverless cache.

### Restoring a backup into a serverless cache (Console)

> **Note**
>
> ElastiCache Serverless supports RDB files compatible with Valkey 7.2 and above, and Redis OSS versions between 5.0 and the latest version available.

#### To restore a backup to a serverless cache (console)

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Backups**.
3. In the list of backups, choose the box to the left of the backup name that you want to restore.
4. Choose **Actions** and then **Restore**.
5. Enter a name for the new serverless cache, and an optional description.
6. Click **Create** to create your new cache and import data from your backup.

### Restoring a backup into a node-based cluster (Console)

#### To restore a backup to a node-based cluster (console)

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Backups**.
3. In the list of backups, choose the box to the left of the backup name you want to restore from.
4. Choose **Actions** and then **Restore**.
5. Choose **Node-based cache** and customize the cluster settings, such as node type, sizes, number of shards, replicas, AZ placement, and security settings.

6. Choose **Create** to create your new node-based cluster and import data from your backup.

### Restoring a backup into a serverless cache (AWS CLI)

> **Note**
>
> ElastiCache Serverless supports RDB files compatible with Valkey 7.2 and above, and Redis OSS versions between 5.0 and the latest version available.

### To restore a backup to a new serverless cache (AWS CLI)

The following AWS CLI example creates a new cache using `create-serverless-cache` and imports data from a backup.

For Linux, macOS, or Unix:

```
aws elasticache create-serverless-cache \

    --serverless-cache-name CacheName \
    --engine redis
    --snapshot-arns-to-restore Snapshot-ARN
```

For Windows:

```
aws elasticache create-serverless-cache ^

    --serverless-cache-name CacheName ^
    --engine redis ^
    --snapshot-arns-to-restore Snapshot-ARN
```

### Restoring a backup into a node-based cluster (AWS CLI)

### To restore a backup to a node-based cluster (AWS CLI)

You can restore a backup to a node-based cluster using the `create-replication-group` operation with the `--snapshot-name` parameter (for an ElastiCache backup) or the `--snapshot-arns` parameter (for an RDB file stored in Amazon S3).

The following example restores from an ElastiCache backup:

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id my-cluster \
    --replication-group-description "Restore from backup" \
    --snapshot-name my-backup
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id my-cluster ^
    --replication-group-description "Restore from backup" ^
    --snapshot-name my-backup
```

The following example restores from an RDB file stored in Amazon S3:

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id my-cluster \
    --replication-group-description "Restore from S3" \
    --snapshot-arns arn:aws:s3:::my-bucket/my-backup.rdb
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id my-cluster ^
    --replication-group-description "Restore from S3" ^
    --snapshot-arns arn:aws:s3:::my-bucket/my-backup.rdb
```

For more information on seeding a new node-based cluster from a backup, see Tutorial: Seeding a new node-based cluster with an externally created backup.

## Deleting a backup

An automatic backup is automatically deleted when its retention limit expires. If you delete a cluster, all of its automatic backups are also deleted. If you delete a replication group, all of the automatic backups from the clusters in that group are also deleted.

ElastiCache provides a deletion API operation that lets you delete a backup at any time, regardless of whether the backup was created automatically or manually. Because manual backups don't have a retention limit, manual deletion is the only way to remove them.

You can delete a backup using the ElastiCache console, the AWS CLI, or the ElastiCache API.

### Deleting a backup (Console)

The following procedure deletes a backup using the ElastiCache console.

**To delete a backup**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Backups**.

   The Backups screen appears with a list of your backups.
3. Choose the box to the left of the name of the backup you want to delete.
4. Choose **Delete**.
5. If you want to delete this backup, choose **Delete** on the **Delete Backup** confirmation screen. The status changes to *deleting*.

### Deleting a serverless backup (AWS CLI)

Use the delete-snapshot AWS CLI operation with the following parameter to delete a serverless backup.

- `--serverless-cache-snapshot-name` – Name of the backup to be deleted.

The following code deletes the backup `myBackup`.

```
aws elasticache delete-serverless-cache-snapshot --serverless-cache-snapshot-
name myBackup
```

For more information, see delete-serverless-cache-snapshot in the *AWS CLI Command Reference*.

### Deleting a node-based cluster backup (AWS CLI)

Use the delete-snapshot AWS CLI operation with the following parameter to delete a node-based cluster backup.

- `--snapshot-name` – Name of the backup to be deleted.

The following code deletes the backup `myBackup`.

```
aws elasticache delete-snapshot --snapshot-name myBackup
```

For more information, see delete-snapshot in the *AWS CLI Command Reference*.

## Tagging backups

You can assign your own metadata to each backup in the form of tags. Tags enable you to categorize your backups in different ways, for example, by purpose, owner, or environment. This is useful when you have many resources of the same type—you can quickly identify a specific resource based on the tags that you've assigned to it. For more information, see Resources you can tag.

Cost allocation tags are a means of tracking your costs across multiple AWS services by grouping your expenses on invoices by tag values. To learn more about cost allocation tags, see Use cost allocation tags.

Using the ElastiCache console, the AWS CLI, or ElastiCache API you can add, list, modify, remove, or copy cost allocation tags on your backups. For more information, see Monitoring costs with cost allocation tags.

# Tutorial: Seeding a new node-based cluster with an externally created backup

When you create a new Valkey or Redis OSS node-based cluster, you can seed it with data from a Valkey or Redis OSS .rdb backup file. Seeding the cluster is useful if you currently manage a Valkey or Redis OSS instance outside of ElastiCache and want to populate your new ElastiCache for Redis OSS node-based cluster with your existing Valkey or Redis OSS data.

To seed a new Valkey or Redis OSS node-based cluster from a Valkey or Redis OSS backup created within Amazon ElastiCache, see Restoring from a backup into a new cache.

When you use a Valkey or Redis OSS .rdb file to seed a new node-based cluster, you can do the following:

- Upgrade from a nonpartitioned cluster to a Valkey or Redis OSS (cluster mode enabled) node-based cluster.
- Specify a number of shards (called node groups in the API and CLI) in the new node-based cluster. This number can be different from the number of shards in the node-based cluster that was used to create the backup file.
- Specify a different node type for the new node-based cluster—larger or smaller than that used in the cluster that made the backup. If you scale to a smaller node type, be sure that the new node type has sufficient memory for your data and Valkey or Redis OSS overhead. For more information, see Ensuring you have enough memory to make a Valkey or Redis OSS snapshot.
- Distribute your keys in the slots of the new Valkey or Redis OSS (cluster mode enabled) cluster differently than in the cluster that was used to create the backup file.

> **Note**
>
> You can't seed a Valkey or Redis OSS (cluster mode disabled) cluster from an .rdb file created from a Valkey or Redis OSS (cluster mode enabled) cluster.

**Important**

- You must ensure that your Valkey or Redis OSS backup data doesn't exceed the resources of the node. For example, you can't upload an .rdb file with 5 GB of Valkey or Redis OSS data to a cache.t3.medium node that has 3.09 GB of memory.

  If the backup is too large, the resulting cluster has a status of `restore-failed`. If this happens, you must delete the cluster and start over.

  For a complete listing of node types and specifications, see Redis OSS node-type specific parameters and Amazon ElastiCache product features and details.
- You can encrypt a Valkey or Redis OSS .rdb file with Amazon S3 server-side encryption using Amazon S3 managed keys (SSE-S3) or AWS KMS keys (SSE-KMS). If you use SSE-KMS, you must grant ElastiCache decrypt permissions on the KMS key and disable Amazon S3 Bucket Keys. For more information, see Grant ElastiCache access to a KMS-encrypted .rdb file. For general information about server-side encryption, see Protecting data using server-side encryption.

Following, you can find topics that walk you through migrating your cluster from outside ElastiCache for Valkey or Redis OSS to ElastiCache for Redis OSS.

**Migrating to ElastiCache for Redis OSS**

- Step 1: Create a Valkey or Redis OSS backup
- Step 2: Create an Amazon S3 bucket and folder
- Step 3: Upload your backup to Amazon S3
- Step 4: Grant ElastiCache read access to the .rdb file

**Migrating from external services to ElastiCache for Redis OSS.**

- Step 1: Create a Valkey or Redis OSS backup
- Step 2: Create an Amazon S3 bucket and folder
- Step 3: Upload your backup to Amazon S3
- Step 4: Grant ElastiCache read access to the .rdb file

## Step 1: Create a Valkey or Redis OSS backup

**To create the Valkey or Redis OSS backup to seed your ElastiCache for Redis OSS instance**

1. Connect to your existing Valkey or Redis OSS instance.
2. Run either BGSAVE or SAVE operation to create a backup. Note where your .rdb file is located.

   BGSAVE is asynchronous and does not block other clients while processing. For more information, see BGSAVE at the Valkey website.

   SAVE is synchronous and blocks other processes until finished. For more information, see SAVE at the Valkey website.

For additional information on creating a backup, see Persistence at the Valkey website.

## Step 2: Create an Amazon S3 bucket and folder

When you have created the backup file, you need to upload it to a folder within an Amazon S3 bucket. To do that, you must first have an Amazon S3 bucket and folder within that bucket. If you already have an Amazon S3 bucket and folder with the appropriate permissions, you can skip to Step 3: Upload your backup to Amazon S3.

**To create an Amazon S3 bucket**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Follow the instructions for creating an Amazon S3 bucket in Creating a bucket in the *Amazon Simple Storage Service User Guide*.

   The name of your Amazon S3 bucket must be DNS-compliant. Otherwise, ElastiCache can't access your backup file. The rules for DNS compliance are:

   - Names must be at least 3 and no more than 63 characters long.
   - Names must be a series of one or more labels separated by a period (.) where each label:
     - Starts with a lowercase letter or a number.
     - Ends with a lowercase letter or a number.
     - Contains only lowercase letters, numbers, and dashes.
   - Names can't be formatted as an IP address (for example, 192.0.2.0).

You must create your Amazon S3 bucket in the same AWS Region as your new ElastiCache for Redis OSS cluster. This approach makes sure that the highest data transfer speed when ElastiCache reads your .rdb file from Amazon S3.

> **Note**
>
> To keep your data as secure as possible, make the permissions on your Amazon S3 bucket as restrictive as you can. At the same time, the permissions still need to allow the bucket and its contents to be used to seed your new Valkey or Redis OSS cluster.

**To add a folder to an Amazon S3 bucket**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the name of the bucket to upload your .rdb file to.
3. Choose **Create folder**.
4. Enter a name for your new folder.
5. Choose **Save**.

   Make note of both the bucket name and the folder name.

## Step 3: Upload your backup to Amazon S3

Now, upload the .rdb file that you created in Step 1: Create a Valkey or Redis OSS backup. You upload it to the Amazon S3 bucket and folder that you created in Step 2: Create an Amazon S3 bucket and folder. For more information on this task, see Add an object to a bucket. Between steps 2 and 3, choose the name of the folder you created .

**To upload your .rdb file to an Amazon S3 folder**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the name of the Amazon S3 bucket you created in Step 2.
3. Choose the name of the folder you created in Step 2.

4. Choose **Upload**.
5. Choose **Add files**.
6. Browse to find the file or files you want to upload, then choose the file or files. To choose multiple files, hold down the Ctrl key while choosing each file name.
7. Choose **Open**.
8. Confirm the correct file or files are listed in the **Upload** dialog box, and then choose **Upload**.

Note the path to your .rdb file. For example, if your bucket name is `myBucket` and the path is `myFolder/redis.rdb`, enter `myBucket/myFolder/redis.rdb`. You need this path to seed the new cluster with the data in this backup.

For additional information, see Bucket restrictions and limitations in the *Amazon Simple Storage Service User Guide*.

## Step 4: Grant ElastiCache read access to the .rdb file

Now, grant ElastiCache read access to your .rdb backup file. You grant ElastiCache access to your backup file in a different way depending if your bucket is in a default AWS Region or an opt-in AWS Region.

AWS Regions introduced before March 20, 2019, are enabled by default. You can begin working in these AWS Regions immediately. Regions introduced after March 20, 2019, such as Asia Pacific (Hong Kong) and Middle East (Bahrain), are disabled by default. You must enable, or opt in, to these Regions before you can use them, as described in Managing AWS regions in *AWS General Reference*.

Choose your approach depending on your AWS Region:

- For a default Region, use the procedure in Grant ElastiCache read access to the .rdb file in a default Region.
- For an opt-in Region, use the procedure in Grant ElastiCache read access to the .rdb file in an opt-in Region.

### Grant ElastiCache read access to the .rdb file in a default Region

AWS Regions introduced before March 20, 2019, are enabled by default. You can begin working in these AWS Regions immediately. Regions introduced after March 20, 2019, such as Asia Pacific

(Hong Kong) and Middle East (Bahrain), are disabled by default. You must enable, or opt in, to these Regions before you can use them, as described in Managing AWS regions in *AWS General Reference*.

**To grant ElastiCache read access to the backup file in an AWS Region enabled by default**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the name of the S3 bucket that contains your .rdb file.
3. Choose the **Permissions** tab.
4. Under **Bucket policy**, choose **Edit**.
5. Add a bucket policy that grants ElastiCache the required permissions. The following example grants the ElastiCache service principal read access to the bucket and its objects. Replace *amzn-s3-demo-bucket* with the name of your S3 bucket. Replace *region-full-name* with the AWS Region identifier where your cluster is located (for example, `us-east-1`).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ElastiCacheSnapshotAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "region-full-name.elasticache-snapshot.amazonaws.com"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketAcl"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

6. Choose **Save changes**.

### Grant ElastiCache read access to the .rdb file in an opt-in Region

AWS Regions introduced before March 20, 2019, are enabled by default. You can begin working in these AWS Regions immediately. Regions introduced after March 20, 2019, such as Asia Pacific (Hong Kong) and Middle East (Bahrain), are disabled by default. You must enable, or opt in, to these Regions before you can use them, as described in Managing AWS regions in *AWS General Reference*.

Now, grant ElastiCache read access to your .rdb backup file.

**To grant ElastiCache read access to the backup file**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the name of the S3 bucket that contains your .rdb file.
3. Choose the name of the folder that contains your .rdb file.
4. Choose the name of your .rdb backup file. The name of the selected file appears above the tabs at the top of the page.
5. Choose the **Permissions** tab.
6. Under **Permissions**, choose **Bucket policy** and then choose **Edit**.
7. Update the policy to grant ElastiCache required permissions to perform operations:

   - Add `[ "Service" : "region-full-name.elasticache-snapshot.amazonaws.com" ]` to `Principal`.
   - Add the following permissions required for exporting a snapshot to the Amazon S3 bucket:
     - `"s3:GetObject"`
     - `"s3:ListBucket"`
     - `"s3:GetBucketAcl"`

   The following is an example of what the updated policy might look like.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "ElastiCacheSnapshotExport",
            "Effect": "Allow",
            "Principal": {
                "Service": "region.elasticache-snapshot.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:ListMultipartUploadParts",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

8. Choose **Save changes**.

### Grant ElastiCache access to a KMS-encrypted .rdb file

If your Amazon S3 bucket uses AWS KMS server-side encryption (SSE-KMS), you must also grant ElastiCache permissions on the KMS key so that it can decrypt the .rdb file during the restore process.

### To grant ElastiCache decrypt permissions on a KMS key

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. Choose the KMS key that is used to encrypt the Amazon S3 bucket.
3. Under **Key policy**, choose **Edit**.
4. Add the following statement to the key policy to grant ElastiCache decrypt permissions:

```
{
    "Sid": "AllowElastiCacheDecrypt",
    "Effect": "Allow",
    "Principal": {
```

```
            "Service": "region-full-name.elasticache-snapshot.amazonaws.com"
        },
        "Action": [
            "kms:Decrypt",
            "kms:DescribeKey"
        ],
        "Resource": "*"
}
```

Replace *region-full-name* with the AWS Region identifier where your cluster is located (for example, us-west-2).

5. Choose **Save changes**.

In addition, you must disable Amazon S3 Bucket Keys for the bucket. ElastiCache does not support reading objects encrypted with Amazon S3 Bucket Keys. To disable this setting, see Configuring S3 Bucket Keys in the *Amazon Simple Storage Service User Guide*.

**Seed the ElastiCache cluster with the .rdb file data**

Now you are ready to create an ElastiCache cluster and seed it with the data from the .rdb file. To create the cluster, follow the directions at Creating a cluster for Valkey or Redis OSS or Creating a Valkey or Redis OSS replication group from scratch. Be sure to choose Valkey or Redis OSS as your cluster engine.

The method you use to tell ElastiCache where to find the backup you uploaded to Amazon S3 depends on the method you use to create the cluster:

**Seed the ElastiCache for Redis OSS cluster or replication group with the .rdb file data**

- **Using the ElastiCache console**

  When selecting **Cluster settings**, choose **Restore from backups** as your cluster creation method, then choose **Other backups** as your **Source** in the **Backup source** section. In the **Seed RDB file S3 location** box, type in the Amazon S3 path for the files(s). If you have multiple .rdb files, type in the path for each file in a comma separated list. The Amazon S3 path looks something like *myBucket*/*myFolder*/*myBackupFilename*.rdb.

- **Using the AWS CLI**

  If you use the create-cache-cluster or the create-replication-group operation, use the parameter --snapshot-arns to specify a fully qualified ARN for each .rdb file. For example,

`arn:aws:s3:::myBucket/myFolder/myBackupFilename.rdb`. The ARN must resolve to the backup files you stored in Amazon S3.

- **Using the ElastiCache API**

  If you use the `CreateCacheCluster` or the `CreateReplicationGroup` ElastiCache API operation, use the parameter `SnapshotArns` to specify a fully qualified ARN for each .rdb file. For example, `arn:aws:s3:::myBucket/myFolder/myBackupFilename.rdb`. The ARN must resolve to the backup files you stored in Amazon S3.

> **Important**
>
> When seeding a Valkey or Redis OSS (cluster mode enabled) cluster, you must configure each node group (shard) in the new cluster or replication group. Use the parameter `--node-group-configuration` (API: `NodeGroupConfiguration`) to do this. For more information, see the following:
>
> - CLI: create-replication-group in the AWS CLI Reference
> - API: CreateReplicationGroup in the ElastiCache API Reference

During the process of creating your cluster, the data in your Valkey or Redis OSS backup is written to the cluster. You can monitor the progress by viewing the ElastiCache event messages. To do this, see the ElastiCache console and choose **Cache Events**. You can also use the AWS ElastiCache command line interface or ElastiCache API to obtain event messages. For more information, see Viewing ElastiCache events.

# Engine versions and upgrading in ElastiCache

This section covers the supported Valkey, Memcached, and Redis OSS engines and how to upgrade. Note that all features available with Redis OSS 7.2 are available in Valkey 7.2 and above by default. You can also upgrade from some existing ElastiCache for Redis OSS engines to a Valkey engine.

## Upgrading engine versions including cross engine upgrades

### Valkey and Redis OSS

With Valkey and Redis OSS, you initiate version upgrades to your cluster or replication group by modifying it using the ElastiCache console, the AWS CLI, or the ElastiCache API and specifying a newer engine version.

You can also cross upgrade from Redis OSS to Valkey. For more information on cross upgrades, see How to upgrade from Redis OSS to Valkey.

### Topics

- How to modify clusters and replication groups
- How to upgrade from Redis OSS to Valkey
- Resolving blocked Valkey or Redis OSS engine upgrades

## How to modify clusters and replication groups

### Valkey and Redis OSS

| Caches | Replication groups |
| --- | --- |
| Using the ElastiCache AWS Management Console | Using the AWS Management Console |
| Using the AWS CLI with ElastiCache | Using the AWS CLI |
| Using the ElastiCache API | Using the ElastiCache API |

### Memcached

With Memcached, to start version upgrades to your cluster, you modify it and specify a newer engine version. You can do this by using the ElastiCache console, the AWS CLI, or the ElastiCache API:

- To use the AWS Management Console, see – Using the ElastiCache AWS Management Console.
- To use the AWS CLI, see Using the AWS CLI with ElastiCache.
- To use the ElastiCache API, see Using the ElastiCache API.

### How to upgrade from Redis OSS to Valkey

Valkey is designed as a drop-in replacement for Redis OSS 7. You can upgrade from Redis OSS to Valkey using the Console, API, or CLI, by specifying the new engine and major engine version. All aspects of your application, including the endpoint DNS name, will remain unchanged, except that for node-based clusters, the underlying node IP addresses will change during the upgrade. You will experience minimal downtime when upgrading from Redis OSS 5.0.6 or higher. The cluster is available for reads during the entire upgrade and is available for writes for most of the upgrade duration, except during the failover operation which lasts a few seconds.

**Note**

**AWS CLI version requirements for Redis OSS to Valkey upgrades:**

- For AWSCLI v1: Minimum required version 1.35.2 (Current version: 1.40.22)
- For AWS CLI v2: Minimum required version 2.18.2 (Current version: 2.27.22)

**Note**

- When upgrading from earlier Redis OSS versions than 5.0.6, you may experience a failover time of 30 to 60 seconds during the DNS propagation.
- To upgrade an existing Redis OSS (cluster mode disabled) single-node cluster to the Valkey engine, first follow these steps: Creating a replication group using an existing cluster. Once the Redis OSS (cluster mode disabled) single-node cluster has been added to a replication group, you can cross-engine upgrade to Valkey.

### Upgrading a replication group from Redis OSS to Valkey

If you have an existing Redis OSS replication group that is using the default cache parameter group, you can upgrade to Valkey by specifying the new engine and engine version with modify-replication-group API.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
   --replication-group-id myReplGroup \
   --engine valkey \
   --engine-version 9.0
```

For Windows:

```
aws elasticache modify-replication-group ^
   --replication-group-id myReplGroup ^
   --engine valkey ^
   --engine-version 9.0
```

If you have a custom cache parameter group applied to the existing Redis OSS replication group you wish to upgrade, you will need to pass a custom Valkey cache parameter group in the request as well. The input Valkey custom parameter group must have the same Redis OSS static parameter values as the existing Redis OSS custom parameter group.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
   --replication-group-id myReplGroup \
   --engine valkey \
   --engine-version 9.0 \
   --cache-parameter-group-name myParamGroup
```

For Windows:

```
aws elasticache modify-replication-group ^
   --replication-group-id myReplGroup ^
   --engine valkey ^
   --engine-version 9.0 ^
   --cache-parameter-group-name myParamGroup
```

### Upgrading a Redis OSS serverless cache to Valkey with the CLI

For Linux, macOS, or Unix:

```
aws elasticache modify-serverless-cache \
   --serverless-cache-name myCluster \
   --engine valkey \
   --major-engine-version 9
```

For Windows:

```
aws elasticache modify-serverless-cache ^
   --serverless-cache-name myCluster ^
   --engine valkey ^
   --major-engine-version 9
```

### Upgrading Redis OSS to Valkey with the Console

#### Upgrading from Redis OSS 5 to Valkey

1. Select the Redis OSS cache to upgrade.
2. An **Upgrade to Valkey** window should appear. Select the **Upgrade to Valkey** button.
3. Go to **Cache settings**, and then select **Engine version**. The most recent version of Valkey is recommended.
4. If this cache is serverless, then you will need to update the parameter group. Go to the **Parameter groups** area of **Cache settings**, select an appropriate parameter group such as *default.valkey8*.
5. Select **Upgrade**.

This cache will now be listed in the Valkey area of the console.

> **Note**
>
> Upgrading directly from Redis OSS 4 or lower to Valkey may include a longer failover time of 30 to 60 seconds during the DNS propagation.

### How to downgrade from Valkey to Redis OSS

If for any reason you wish to rollback your upgraded cluster, Amazon ElastiCache supports rolling back a Valkey 7.2 cache to Redis OSS 7.1. You can perform a rollback using the same console, API, or CLI steps as an engine upgrade by specifying Redis OSS 7.1 as the target engine version. All aspects of your application, including the endpoint DNS name, will remain unchanged, except that for node-based clusters, the underlying node IP addresses will change during the rollback. You will experience minimal downtime while rolling back. The cluster is available for reads during the entire rollback and is available for writes for most of the rollback duration, except during the failover operation which lasts a few seconds.

Additionally, you can restore a snapshot created from your Valkey 7.2 cache as a Redis OSS 7.1 cache. When you restore from a snapshot, you can specify Redis OSS 7.1 as the target engine version. When using this option, a new cache will be created from the snapshot. Restoring from a snapshot has no effect on the Valkey cache that the snapshot was created from.

The following requirements and limitations apply when performing a rollback:

- ElastiCache only supports rolling back from Valkey 7.2 to Redis OSS 7.1. This is true even if you upgraded to Valkey 7.2 from an earlier version than Redis OSS 7.1.
- Any user group and user associated with the replication group or serverless cache being rolled back must be configured with engine type `REDIS`.

## Resolving blocked Valkey or Redis OSS engine upgrades

As shown in the following table, your Valkey or Redis OSS engine upgrade operation is blocked if you have a pending scale up operation.

| Pending operations | Blocked operations |
| --- | --- |
| Scale up | Immediate engine upgrade |
| Engine upgrade | Immediate scale up |
| Scale up and engine upgrade | Immediate scale up |
| | Immediate engine upgrade |

**To resolve a blocked Valkey or Redis OSS engine upgrade**

- Do one of the following:
  - Schedule your Redis OSS or Valkey engine upgrade operation for the next maintenance window by clearing the **Apply immediately** check box.

    With the CLI, use `--no-apply-immediately`. With the API, use `ApplyImmediately=false`.
  - Wait until your next maintenance window (or after) to perform your Redis OSS engine upgrade operation.
  - Add the Redis OSS scale up operation to this cluster modification with the **Apply Immediately** check box chosen.

    With the CLI, use `--apply-immediately`. With the API, use `ApplyImmediately=true`.

    This approach effectively cancels the engine upgrade during the next maintenance window by performing it immediately.

## ElastiCache Extended Support

With ElastiCache Extended Support, you can continue running your cache on a major engine version past the end of standard support date for an additional cost. If you don't upgrade after the end of standard support date, you will be charged.

Extended Support provides the following updates and technical support:

- Security updates for critical and high CVEs for your cache and cache engine
- Bug fixes and patches for critical issues
- The ability to open support cases and receive troubleshooting help within the standard ElastiCache service level agreement

This paid offering gives you more time to upgrade to a supported major engine version.

For example, the ElastiCache end of standard support date for Redis OSS 4.0.10 is January 31, 2026. If you aren't ready to manually upgrade to Valkey or to Redis OSS 6 or later by that date, ElastiCache will automatically enroll your caches in Extended Support and you can continue to run

Redis OSS 4.0.10. Starting the first day of the month after the standard support ends, February 1, 2026, ElastiCache automatically charges you for Extended Support.

Extended Support is available for up to 3 years past the end of standard support date for a major engine version. For Elasticache for Redis OSS versions 4 and 5, that will be January 31, 2029. After this date, any caches still running Redis OSS versions 4 and 5 will be automatically upgraded to the latest version of Valkey.

Once an engine's support period ends, caches that continue to run that old version will automatically transition to Extended Support. You will be notified before the Extended Support pricing start date so you may upgrade your instance instead. You also can explicitly opt out at any time by upgrading to supported versions.

For more information about the end of standard support dates and the end of Extended Support dates, see ElastiCache versions for Redis OSS end of life schedule for Valkey, Memcached or Redis OSS.

**Topics**


- ElastiCache Extended Support charges
- Versions with ElastiCache Extended Support
- ElastiCache and customer responsibilities with ElastiCache Extended Support


## ElastiCache Extended Support charges

You will incur charges for all engines enrolled in ElastiCache Extended Support beginning the day after the end of standard support. For the ElastiCache end of standard support date, see Versions with ElastiCache Extended Support.

The additional charge for ElastiCache Extended Support automatically stops when you take one of the following actions:

- Upgrade to an engine version that's covered under standard support.
- Delete the cache that's running a major version past the ElastiCache end of standard support date.

The charges will restart if your target engine version enters Extended Support in the future.

For example, let's say ElastiCache version 4 for Redis OSS enters Extended Support on February 1, 2026, and you upgrade your caches on v4 to v6 on January 1, 2027. You will only be charged for 11 months of Extended Support, on ElastiCache version 4 for Redis OSS. If you continue running ElastiCache version 6 for Redis OSS past its end of standard support date of January 31, 2027, then those caches will again incur Extended Support charges starting on February 1, 2027.

You can avoid being charged for ElastiCache Extended Support by preventing ElastiCache from creating or restoring a cache past the ElastiCache end of standard support date.

**Extended Support charges and reserved nodes**

ElastiCache reserved nodes or instances (RIs) pricing does not discount ElastiCache Extended Support charges. If you have a reserved node or instance, you are billed for:

- The standard ElastiCache node price with the RI discount applied.
- The ElastiCache Extended Support charge, calculated as a percentage of the standard on-demand node price. The RI discount does not apply to Extended Support charges.

For example, if you are running a cache.m5.large node on ElastiCache version 5 for Redis OSS in the US East (Ohio) Region, the on-demand price is $0.156/hr and the Extended Support premium for Year 1 is 80%. If your RI effective rate is $0.106/hr, you pay the RI-discounted rate of $0.106/hr plus the Extended Support charge of $0.1248/hr (80% of the $0.156/hr on-demand price), for a total of $0.2308/hr per node.

For more information, see Amazon ElastiCache pricing.

## Versions with ElastiCache Extended Support

Redis Open Source Software (OSS) versions 4 and 5 reached their community End of Life in 2020 and 2022, respectively. This means no further updates, bug fixes, or security patches are being released by the community. Standard support for ElastiCache Redis OSS versions 4 and 5 on ElastiCache will end on January 31, 2026. Continuing to use unsupported versions of Redis OSS could leave your data vulnerable to known Common Vulnerabilities and Exposures (CVEs).

Starting on February 1, 2026, ElastiCache caches still running on Redis OSS versions 4 and 5 will be automatically enrolled in Extended Support, to provide continuous availability and security. Although Extended Support offers flexibility, we recommend treating the end of standard support as a planning milestone for your production workloads. We strongly encourage you to upgrade

your Redis OSS v4 and v5 caches to ElastiCache for Valkey or Redis OSS v6 or later, before the end of standard support.

The following table summarizes the Amazon ElastiCache end of standard support date and Extended Support dates.

**Extended support and End of Life schedule**

| Major Engine Version | End of Standard Support | Start of Extended Support Y1 Premium | Start of Extended Support Y2 Premium | Start of Extended Support Y3 Premium | End of Extended Support and version EOL |
|---|---|---|---|---|---|
| Redis OSS v4 | 1/31/2026 | 2/1/2026 | 2/1/2027 | 2/1/2028 | 1/31/2029 |
| Redis OSS v5 | 1/31/2026 | 2/1/2026 | 2/1/2027 | 2/1/2028 | 1/31/2029 |
| Redis OSS v6 | 1/31/2027 | 2/1/2027 | 2/1/2028 | 2/1/2029 | 1/31/2030 |

Extended Support will only be offered for the latest supported patch version of each major Redis OSS version. When Extended Support begins on February 1, 2026, if your Redis OSS v4 and v5 clusters are not already on the latest patch versions, they will be automatically upgraded to v4.0.10 for Redis OSS v4, and v5.0.6 for Redis OSS v5, before being enrolled in Extended Support. This ensures that you receive security updates and bug fixes through Extended Support. You do not need to take any action to upgrade to these latest patch versions as part of the Extended Support transition.

## ElastiCache and customer responsibilities with ElastiCache Extended Support

Following are the responsibilities of Amazon ElastiCache, and your responsibilities with ElastiCache Extended Support.

**Amazon ElastiCache responsibilities**

After the ElastiCache end of standard support date, Amazon ElastiCache will supply patches, bug fixes, and upgrades for engines that are enrolled in ElastiCache Extended Support. This will occur for up to 3 years, or until you stop using the engines in Extended Support, whichever happens first.

**Your responsibilities**

You're responsible for applying the patches, bug fixes, and upgrades given for caches in ElastiCache Extended Support. Amazon ElastiCache reserves the right to change, replace, or withdraw such patches, bug fixes, and upgrades at any time. If a patch is necessary to address security or critical stability issues, Amazon ElastiCache reserves the right to update your caches with the patch, or to require that you install the patch.

You're also responsible for upgrading your engine to a newer engine version before the ElastiCache end of Extended Support date. The ElastiCache end of Extended Support date is typically 3 years after the ElastiCache end of standard support date.

If you don't upgrade your engine, then after the ElastiCache end of Extended Support date, Amazon ElastiCache will attempt to upgrade your engine to a newer engine version that's supported under ElastiCache standard support. If the upgrade fails, then Amazon ElastiCache reserves the right to delete the cache that's running the engine past the ElastiCache end of standard support date. However, before doing so, Amazon ElastiCache will preserve your data from that engine.

# Version Management for ElastiCache

Manage how you would like to update your ElastiCache caches and node-based clusters updated for the Valkey, Memcached, and Redis OSS engines.

## Version management for ElastiCache Serverless Cache

Manage if and when the ElastiCache Serverless cache is upgraded and perform version upgrades on your own terms and timelines.

ElastiCache Serverless automatically applies the latest minor and patch software version to your cache, without any impact or downtime to your application. No action is required on your end.

When a new major version is available, ElastiCache Serverless will send you a notification in the console and an event in EventBridge. You can choose to upgrade your cache to the latest major version by modifying your cache using the Console, CLI, or API and selecting the latest engine version. Similar to minor and patch upgrades, major version upgrades are performed without downtime to your application.

## Version management for node-based ElastiCache clusters

When working with node-based ElastiCache clusters, you can control when the software powering your cluster is upgraded to new versions that are supported by ElastiCache. You can control when

to upgrade your cache to the latest available major, minor, and patch versions. You initiate engine version upgrades to your cluster or replication group by modifying it and specifying a new engine version.

You can control if and when the protocol-compliant software powering your cluster is upgraded to new versions that are supported by ElastiCache. This level of control enables you to maintain compatibility with specific versions, test new versions with your application before deploying in production, and perform version upgrades on your own terms and timelines.

Because version upgrades might involve some compatibility risk, they don't occur automatically. You must initiate them.

**Valkey and Redis OSS clusters**

**Note**

- If a Valkey or Redis OSS cluster is replicated across one or more Regions, the engine version is upgraded for secondary Regions and then for the primary Region.
- ElastiCache for Redis OSS versions are identified with a semantic version which comprise a major and minor component. For example, in Redis OSS 6.2, the major version is 6, and the minor version 2. When operating node-based clusters, ElastiCache for Redis OSS also exposes the patch component, e.g. Redis OSS 6.2.1, and the patch version is 1.

  Major versions are for API incompatible changes and minor versions are for new functionality added in a backwards-compatible way. Patch versions are for backwards-compatible bug fixes and non-functional changes.

With Valkey and Redis OSS, you initiate engine version upgrades to your cluster or replication group by modifying it and specifying a new engine version. For more information, see Modifying a replication group.

**Memcached**

With Memcached, to upgrade to a newer version you must modify your cluster and specify the new engine version you want to use. Upgrading to a newer Memcached version is a destructive process – you lose your data and start with a cold cache. For more information, see Modifying an ElastiCache cluster.

You should be aware of the following requirements when upgrading from an older version of Memcached to Memcached version 1.4.33 or newer. `CreateCacheCluster` and `ModifyCacheCluster` fails under the following conditions:

- If `slab_chunk_max > max_item_size`.
- If `max_item_size modulo slab_chunk_max != 0`.
- If `max_item_size > ((max_cache_memory - memcached_connections_overhead) / 4)`.

  The value `(max_cache_memory - memcached_connections_overhead)` is the node's memory useable for data. For more information, see Memcached connection overhead.

## ElastiCache version 9.1 for Valkey

Here are some of the new features introduced in Valkey 9.1 (compared to ElastiCache Valkey 9.0):

- **Up to 20% memory reduction for common string patterns** – Strings under 128 bytes now use 20% less memory. Sorted sets see up to a 10% memory reduction. Bulk delete operations (SREM, ZREM, HDEL) pause auto-shrink during bulk ops for faster execution.
- **HGETDEL, MSETEX, and CLUSTERSCAN commands** – Three new commands: `HGETDEL` atomically retrieves and deletes hash fields, `MSETEX` sets multiple keys with a shared expiration in a single call, and `CLUSTERSCAN` provides cluster-wide key scanning across all nodes.
- **Database-level ACL access control** – Per-database permissions via `ACL SETUSER ... db=0,1` syntax give you fine-grained access control at the database level for multi-tenant workloads on a single cluster.
- Over 70 additional enhancements including reduced rehashing latency spikes, CLUSTER SHARDS availability-zone field, HSETEX NX/XX flags, hardware clock enabled by default, JSON logging format, and main/IO thread cumulative usage metrics through INFO ALL. For the complete list, see Valkey 9.1 Release Notes.

For more information on Valkey, see Valkey.

For more information on the Valkey 9.1 release, see Valkey 9.1 Release Notes.

## ElastiCache version 9.0 for Valkey

Here are some of the new features introduced in Valkey 9.0 (compared to ElastiCache Valkey 8.2):

- **Durability** – ElastiCache version 9.0 for Valkey introduces durability via a Multi-AZ transactional log. Choose between synchronous writes (zero data loss, single-digit millisecond write latency) or asynchronous writes (microsecond write latency, up to 10 seconds of data at risk). Durability enables ElastiCache to serve as a data store for system-of-record workloads.
- **Public endpoints for ElastiCache Serverless**. Connect to your cache over the internet without a VPC. Public endpoints require IAM authentication and TLS 1.3. Available on ElastiCache Serverless only.
- **Full-text search, aggregations, and hybrid search** – ElastiCache version 9.0 for Valkey extends the search capabilities introduced in Valkey 8.2 with full-text search, aggregation pipelines, and hybrid queries that combine text and vector results. Build powerful search experiences directly in your cache without managing a separate search engine. These capabilities are based on the open-source valkey-search project on the GitHub website. Available on ElastiCache Serverless with Valkey 9.0 and later and on node-based clusters.
- **Hash field expiration** – Set TTLs on individual fields within a hash, giving you fine-grained control over data lifecycle without managing separate keys. Expire stale fields automatically while keeping the rest of the hash intact. (#2089)
- **Multi-database support in cluster mode** – Use numbered databases (`SELECT 0`–15) in cluster-mode-enabled configurations, removing a long-standing limitation and simplifying migrations from standalone deployments to horizontally scaled clusters. (#1671)
- **Up to 40% higher throughput with pipelining** – Optimized command parsing and prefetching across pipelined requests delivers significantly higher throughput for batch-heavy workloads, so you can process more operations per second without adding nodes. (#2092)
- **Polygon-based geospatial queries** – The new `BYPOLYGON` option for `GEOSEARCH` and `GEOSEARCHSTORE` lets you query members inside arbitrary polygon boundaries, enabling real-world geofencing use cases like delivery zones, service areas, and regional targeting. (#1809)
- Over 100 additional enhancements across the 9.0.0 through 9.0.3 releases, including 13 new features, 16 performance optimizations with SIMD acceleration, support for DELIFEQ, expanded observability with per-node and per-slot metrics, MPTCP networking support, and automatic TLS certificate-based authentication. For the complete list, see Valkey 9.0 Release Notes.

For more information on Valkey, see Valkey.

For more information on the Valkey 9.0 release, see Valkey 9.0 Release Notes.

## ElastiCache version 8.2 for Valkey

Here are some of the new features introduced in Valkey 8.2 (compared to ElastiCache Valkey 8.1):

- ElastiCache for Valkey v8.2 provides native support for vector search, delivering latency as low as microseconds-the lowest latency vector search with the highest throughput and best price-performance at 95%+ recall rate among popular vector databases on AWS.

For more information on Valkey, see Valkey.

ElastiCache version 8.2 for Valkey enhances Valkey 8.1 with vector search capabilities based on valkey-search module. For more information on the Valkey 8.2 release, see release notes for valkey-search. Please note that ElastiCache v8.2 is compatible with Valkey v8.1.

## ElastiCache version 8.1 for Valkey

Here are some of the new features introduced in Valkey 8.1 (compared to ElastiCache Valkey 8.0):

- A new hash table implementation that reduces memory overhead to lower memory usage by as much as 20% for common key/value patterns.
- Native support for Bloom filters, a new data type allowing you to perform lookups using as much as 98% less memory compared to using the Set data type.
- New command COMMANDLOG that records slow executions, large requests, and large replies.
- New conditional update support to the SET command using IFEQ argument.
- Performance improvements, including up to 45% lower latency for the ZRANK command, up to 12x faster performance for PFMERGE and PFCOUNT, and up to 514% higher throughput for BITCOUNT.

For more information on Valkey, see Valkey

For more information on the Valkey 8.1 release, see Valkey 8.1 Release Notes

## ElastiCache version 8.0 for Valkey

Here are some of the new features introduced in Valkey 8.0 (compared to ElastiCache Valkey 7.2.6):

- Memory efficiency improvements, allowing users to store up to 20% more data per node without any application changes.

- Newly-introduced per-slot metrics infrastructure for node-based clusters, providing detailed visibility into the performance and resource usage of individual slots.
- ElastiCache Serverless for Valkey 8.0 can double the supported requests per second (RPS) every 2-3 minutes, reaching 5M RPS per cache from zero in under 13 minutes, with consistent sub-millisecond p50 read latency.

For more information on Valkey, see Valkey

For more information on the Valkey 8 release, see Valkey 8 Release Notes

## ElastiCache version 7.2.6 for Valkey

On October 10 2024, ElastiCache version 7.2.6 for Valkey was released. Here are some of the new features introduced in 7.2 (compared to ElastiCache version 7.1 for Redis OSS):

- Performance and memory optimizations for various data types: memory optimization for list and set type keys, speed optimization for sorted sets commands, performance optimization for commands with multiple keys in cluster mode, pub/sub performance improvements, performance optimization for SCAN, SSCAN, HSCAN, ZSCAN commands and numerous other smaller optimizations.
- New WITHSCORE option for ZRANK and ZREVRANK commands
- CLIENT NO-TOUCH for clients to run commands without affecting LRU/LFU of keys.
- New command CLUSTER MYSHARDID that returns the Shard ID of the node to logically group nodes in cluster mode based on replication.

For more information on Valkey, see Valkey

For more information on the ElastiCache version 7.2 for Valkey release, see Redis OSS 7.2.4 Release Notes (ElastiCache version 7.2 for Valkey includes all changes from ElastiCache version 7.1 for Redis OSS up to ElastiCache version 7.2.4 for Redis OSS). Valkey 7.2 release notes at Valkey on GitHub.

## Supported Redis OSS engine versions

ElastiCache Serverless caches and node-based clusters support all Redis OSS versions 7.1 and before.

- ElastiCache version 7.1 for Redis OSS (enhanced)

**Node-based ElastiCache clusters support the following Redis OSS versions:**

- ElastiCache version 7.1 for Redis OSS (enhanced)
- ElastiCache version 7.0 for Redis OSS (enhanced)
- ElastiCache version 6.2 for Redis OSS (enhanced)
- ElastiCache version 6.0 for Redis OSS (enhanced)
- ElastiCache version 5.0.6 for Redis OSS (enhanced)
- ElastiCache version 5.0.5 for Redis OSS (deprecated, use version 5.0.6)
- ElastiCache version 5.0.4 for Redis OSS (deprecated, use version 5.0.6)
- ElastiCache version 5.0.3 for Redis OSS (deprecated, use version 5.0.6)
- ElastiCache version 5.0.0 for Redis OSS (deprecated, use version 5.0.6)
- ElastiCache version 4.0.10 for Redis OSS (enhanced)
- Past End of Life (EOL) versions (3.x)
- Past End of Life (EOL) versions (2.x)

## ElastiCache version 7.1 for Redis OSS (enhanced)

This release contains performance improvements which enable workloads to drive higher throughput and lower operation latencies. ElastiCache version 7.1 for Redis OSS introduces two main enhancements :

We extended the enhanced I/O threads functionality to also handle the presentation layer logic. By presentation layer, we mean the Enhanced I/O threads which are now not only reading client input, but also parsing the input into Redis OSS binary command format. This is then forwarded to the main thread for execution which provides performance gain. Improved Redis OSS memory access pattern. Execution steps from many data structure operations are interleaved, to ensure parallel memory access and reduced memory access latency. When running ElastiCache on Graviton3-based `R7g.4xlarge` or larger, customers can achieve over 1 million requests per second per node. With the performance improvements to ElastiCache for Redis OSS v7.1, customers can achieve up to 100% more throughput and 50% lower P99 latency relative to ElastiCache for Redis OSS v7.0. These enhancements are enabled on node sizes with at least 8 physical cores (`2xlarge` on Graviton, and `4xlarge` on x86), regardless of the CPU type and require no client changes.

> **Note**
>
> ElastiCache v7.1 is compatible with Redis OSS v7.0.

## ElastiCache version 7.0 for Redis OSS (enhanced)

ElastiCache for Redis OSS 7.0 adds a number of improvements and support for new functionality:

- Functions: ElastiCache for Redis OSS 7 adds support for Redis OSS Functions, and provides a managed experience enabling developers to execute LUA scripts with application logic stored on the ElastiCache cluster, without requiring clients to re-send the scripts to the server with every connection.
- ACL improvements: Valkey and Redis OSS 7 adds support for the next version of Access Control Lists (ACLs). Clients can now specify multiple sets of permissions on specific keys or keyspaces in Valkey and Redis OSS.
- Sharded Pub/Sub: ElastiCache for Valkey and Redis OSS 7 adds support to run Pub/Sub functionality in a sharded way when running ElastiCache in Cluster Mode Enabled (CME). Pub/Sub capabilities enable publishers to issue messages to any number of subscribers on a channel. Channels are bound to a shard in the ElastiCache cluster, eliminating the need to propagate channel information across shards resulting in improved scalability.
- Enhanced I/O multiplexing: ElastiCache for Valkey and Redis OSS 7 introduces enhanced I/O multiplexing, which delivers increased throughput and reduced latency for high-throughput workloads that have many concurrent client connections to an ElastiCache cluster. For example, when using a cluster of r6g.xlarge nodes and running 5200 concurrent clients, you can achieve up to 72% increased throughput (read and write operations per second) and up to 71% decreaseed P99 latency, compared with ElastiCache version 6 for Redis OSS.

For more information on Valkey, see Valkey. For more information on the Redis OSS 7.0 release, see Redis OSS 7.0 Release Notes at Redis OSS on GitHub.

## ElastiCache version 6.2 for Redis OSS (enhanced)

ElastiCache for Redis OSS 6.2 includes performance improvements for TLS-enabled clusters using x86 node types with 8 vCPUs or more or Graviton2 node types with 4 vCPUs or more. These enhancements improve throughput and reduce client connection establishment time by offloading

encryption to other vCPUs. With Redis OSS 6.2, you can also manage access to Pub/Sub channels with Access Control List (ACL) rules.

With this version, we also introduce support for data tiering on cluster nodes containing locally attached NVMe SSD. For more information, see Data tiering in ElastiCache.

Redis OSS engine version 6.2.6 also introduces support for native JavaScript Object Notation (JSON) format, a simple, schemaless way to encode complex datasets inside Redis OSS clusters. With JSON support, you can leverage the performance and Redis OSS APIs for applications that operate over JSON. For more information, see Getting started with JSON. Also included are JSON-related metrics, `JsonBasedCmds` and `JsonBasedCmdsLatency`, that are incorporated into CloudWatch to monitor the usage of this datatype. For more information, see Metrics for Valkey and Redis OSS.

You specify the engine version by using 6.2. ElastiCache will automatically invoke the preferred patch version of Redis OSS 6.2 that is available. For example, when you create/modify a cluster, you set the `--engine-version` parameter to 6.2. The cluster will be launched with the current available preferred patch version of Redis OSS 6.2 at the creation/modification time. Specifying engine version 6.x in the API will result in the latest minor version of Redis OSS 6.

For existing 6.0 clusters, you can opt-in to the next auto minor version upgrade by setting the `AutoMinorVersionUpgrade` parameter to `yes` in the `CreateCacheCluster`, `ModifyCacheCluster`, `CreateReplicationGroup` or `ModifyReplicationGroup` APIs. ElastiCache will upgrade the minor version of your existing 6.0 clusters to 6.2 using self-service updates. For more information, see Self-service updates in Amazon ElastiCache.

When calling the DescribeCacheEngineVersions API, the `EngineVersion` parameter value will be set to 6.2 and the actual engine version with the patch version will be returned in the `CacheEngineVersionDescription` field.

For more information on the Redis OSS 6.2 release, see Redis OSS 6.2 Release Notes at Redis OSS on GitHub.

## ElastiCache version 6.0 for Redis OSS (enhanced)

Amazon ElastiCache introduces the next version of ElastiCache for the Redis OSS engine, which includes Authenticating Users with Role Based Access Control, client-side caching and significant operational improvements.

Beginning with Redis OSS 6.0, ElastiCache will offer a single version for each Redis OSS minor release, rather than offering multiple patch versions. ElastiCache will automatically manage the patch version of your running clusters, ensuring improved performance and enhanced security.

You can also opt-in to the next auto minor version upgrade by setting the `AutoMinorVersionUpgrade` parameter to `yes` and ElastiCache will manage the minor version upgrade, through self-service updates. For more information, see Service updates in ElastiCache.

You specify the engine version by using `6.0`. ElastiCache will automatically invoke the preferred patch version of Redis OSS 6.0 that is available. For example, when you create/modify a cluster, you set the `--engine-version` parameter to 6.0. The cluster will be launched with the current available preferred patch version of Redis OSS 6.0 at the creation/modification time. Any request with a specific patch version value will be rejected, an exception will be thrown and the process will fail.

When calling the DescribeCacheEngineVersions API, the `EngineVersion` parameter value will be set to 6.0 and the actual engine version with the patch version will be returned in the `CacheEngineVersionDescription` field.

For more information on the Redis OSS 6.0 release, see Redis OSS 6.0 Release Notes at Redis OSS on GitHub.

### ElastiCache version 5.0.6 for Redis OSS (enhanced)

Amazon ElastiCache introduces the next version of ElastiCache for the Redis OSS engine, which includes bug fixes and the following cumulative updates:

- Engine stability guarantee in special conditions.
- Improved Hyperloglog error handling.
- Enhanced handshake commands for reliable replication.
- Consistent message delivery tracking via `XCLAIM` command.
- Improved `LFU` field management in objects.
- Enhanced transaction management when using `ZPOP`.
- Ability to rename commands: A parameter called `rename-commands` that allows you to rename potentially dangerous or expensive Redis OSS commands that might cause accidental data loss, such as `FLUSHALL` or `FLUSHDB`. This is similar to the rename-command configuration in open source Redis OSS. However, ElastiCache has improved the experience by providing a fully managed workflow. The command name changes are applied immediately, and

automatically propagated across all nodes in the cluster that contain the command list. There is no intervention required on your part, such as rebooting nodes.

The following examples demonstrate how to modify existing parameter groups. They include the `rename-commands` parameter, which is a space-separated list of commands you want to rename:

```
aws elasticache modify-cache-parameter-group --cache-parameter-group-
name custom_param_group
--parameter-name-values "ParameterName=rename-commands,  ParameterValue='flushall
 restrictedflushall'" --region region
```

In this example, the *rename-commands* parameter is used to rename the `flushall` command to `restrictedflushall`.

To rename multiple commands, use the following:

```
aws elasticache modify-cache-parameter-group --cache-parameter-group-
name custom_param_group
--parameter-name-values "ParameterName=rename-commands,  ParameterValue='flushall
 restrictedflushall flushdb restrictedflushdb'" --region region
```

To revert any change, re-run the command and exclude any renamed values from the `ParameterValue` list that you want to retain, as shown following:

```
aws elasticache modify-cache-parameter-group --cache-parameter-group-
name custom_param_group
--parameter-name-values "ParameterName=rename-commands,  ParameterValue='flushall
 restrictedflushall'" --region region
```

In this case, the `flushall` command is renamed to `restrictedflushall` and any other renamed commands revert to their original command names.

> **Note**
>
> When renaming commands, you are restricted to the following limitations:
>
> - All renamed commands should be alphanumeric.
> - The maximum length of new command names is 20 alphanumeric characters.

- When renaming commands, ensure that you update the parameter group associated with your cluster.
- To prevent a command's use entirely, use the keyword `blocked`, as shown following:

```
aws elasticache modify-cache-parameter-group --cache-parameter-group-
name custom_param_group
--parameter-name-values "ParameterName=rename-commands,
 ParameterValue='flushall blocked'" --region region
```

For more information on the parameter changes and a list of what commands are eligible for renaming, see Redis OSS 5.0.3 parameter changes.

- Redis OSS Streams: This models a log data structure that allows producers to append new items in real time. It also allows consumers to consume messages either in a blocking or nonblocking fashion. Streams also allow consumer groups, which represent a group of clients to cooperatively consume different portions of the same stream of messages, similar to Apache Kafka. For more information, see Streams.
- Support for a family of stream commands, such as `XADD`, `XRANGE` and `XREAD`. For more information, see Streams Commands.
- A number of new and renamed parameters. For more information, see Redis OSS 5.0.0 parameter changes.
- A new Redis OSS metric, `StreamBasedCmds`.
- Slightly faster snapshot time for Redis OSS nodes.

**Important**

ElastiCache has back-ported two critical bug fixes from Redis OSS open source version 5.0.1. They are listed following:

- RESTORE mismatch reply when certain keys have already expired.
- The `XCLAIM` command can potentially return a wrong entry or desynchronize the protocol.

Both of these bug fixes are included in ElastiCache for Redis OSS support for Redis OSS engine version 5.0.0 and are consumed in future version updates.

For more information, see Redis OSS 5.0.6 Release Notes at Redis OSS on GitHub.

## ElastiCache version 5.0.5 for Redis OSS (deprecated, use version 5.0.6)

Amazon ElastiCache introduces the next version of ElastiCache for the Redis OSS engine;. It includes online configuration changes for ElastiCache of auto-failover clusters during all planned operations. You can now scale your cluster, upgrade the Redis OSS engine version and apply patches and maintenance updates while the cluster stays online and continues serving incoming requests. It also includes bug fixes.

For more information, see Redis OSS 5.0.5 Release Notes at Redis OSS on GitHub.

## ElastiCache version 5.0.4 for Redis OSS (deprecated, use version 5.0.6)

Amazon ElastiCache introduces the next version of the Redis OSS engine supported by ElastiCache. It includes the following enhancements:

- Engine stability guarantee in special conditions.
- Improved Hyperloglog error handling.
- Enhanced handshake commands for reliable replication.
- Consistent message delivery tracking via `XCLAIM` command.
- Improved `LFU` field management in objects.
- Enhanced transaction management when using `ZPOP`.

For more information, see Redis OSS 5.0.4 Release Notes at Redis OSS on GitHub.

## ElastiCache version 5.0.3 for Redis OSS (deprecated, use version 5.0.6)

Amazon ElastiCache introduces the next version of ElastiCache for the Redis OSS engine, which includes bug fixes.

## ElastiCache version 5.0.0 for Redis OSS (deprecated, use version 5.0.6)

Amazon ElastiCache introduces the next major version of ElastiCache for the Redis OSS engine. ElastiCache version 5.0.0 for Redis OSS brings support for the following improvements:

- Redis OSS Streams: This models a log data structure that allows producers to append new items in real time. It also allows consumers to consume messages either in a blocking or nonblocking fashion. Streams also allow consumer groups, which represent a group of clients to cooperatively consume different portions of the same stream of messages, similar to Apache Kafka. For more information, see Streams.
- Support for a family of stream commands, such as `XADD`, `XRANGE` and `XREAD`. For more information, see Streams Commands.
- A number of new and renamed parameters. For more information, see Redis OSS 5.0.0 parameter changes.
- A new Redis OSS metric, `StreamBasedCmds`.
- Slightly faster snapshot time for Redis OSS nodes.

## ElastiCache version 4.0.10 for Redis OSS (enhanced)

Amazon ElastiCache introduces the next major version of ElastiCache for the Redis OSS engine. ElastiCache version 4.0.10 for Redis OSS brings support for the following improvements:

- Both online cluster resizing and encryption in a single ElastiCache version. For more information, see the following:
  - Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters
  - Online resharding for Valkey or Redis OSS (cluster mode enabled)
  - Data security in Amazon ElastiCache
- A number of new parameters. For more information, see Redis OSS 4.0.10 parameter changes.
- Support for family of memory commands, such as `MEMORY`. For more information, see Commands (search on MEMO).
- Support for memory defragmentation while online thus allowing more efficient memory utilization and more memory available for your data.
- Support for asynchronous flushes and deletes. ElastiCache for Redis OSS supports commands like `UNLINK`, `FLUSHDB` and `FLUSHALL` to run in a different thread from the main thread. Doing

this helps improve performance and response times for your applications by freeing memory asynchronously.

- A new Redis OSS metric, `ActiveDefragHits`. For more information, see Metrics for Redis OSS.

Redis OSS (cluster mode disabled) users running ElastiCache version 3.2.10 for Redis OSS can use the console to upgrade their clusters via online upgrade.

**Comparing ElastiCache cluster resizing and encryption support**

| Feature | 3.2.6 | 3.2.10 | 4.0.10 and later |
|---|---|---|---|
| Online cluster resizing * | No | Yes | Yes |
| In-transit encryption ** | Yes | No | Yes |
| At rest encryption ** | Yes | No | Yes |
| * Adding, removing, and rebalancing shards. | | | |
| ** Required for FedRAMP, HIPAA, and PCI DSS compliant applications. For more information, see Compliance validation for Amazon ElastiCache. | | | |

## Past End of Life (EOL) versions (3.x)

### ElastiCache version 3.2.10 for Redis OSS (enhanced)

Amazon ElastiCache introduces the next major version of ElastiCache for the Redis OSS engine. ElastiCache version 3.2.10 for Redis OSS (enchanced) introduces online cluster resizing to add or remove shards from the cluster while it continues to serve incoming I/O requests. ElastiCache for Redis OSS 3.2.10 users have all the functionality of earlier Redis OSS versions except the ability to encrypt their data. This ability is currently available only in version 3.2.6.

**Comparing ElastiCache versions 3.2.6 and 3.2.10 for Redis OSS**

| Feature | 3.2.6 | 3.2.10 |
|---|---|---|
| Online cluster resizing * | No | Yes |
| In-transit encryption ** | Yes | No |

| Feature | 3.2.6 | 3.2.10 |
|---|---|---|
| At rest encryption ** | Yes | No |
| * Adding, removing, and rebalancing shards. | | |
| ** Required for FedRAMP, HIPAA, and PCI DSS compliant applications. For more information, see Compliance validation for Amazon ElastiCache. | | |

For more information, see the following:

- Online resharding for Valkey or Redis OSS (cluster mode enabled)
- Online cluster resizing

### ElastiCache version 3.2.6 for Redis OSS (enhanced)

Amazon ElastiCache introduces the next major version of ElastiCache for the Redis OSS engine. ElastiCache version 3.2.6 for Redis OSS users have access to all the functionality of earlier Redis OSS versions, plus the option to encrypt their data. For more information, see the following:

- ElastiCache in-transit encryption (TLS)
- At-Rest Encryption in ElastiCache
- Compliance validation for Amazon ElastiCache

### ElastiCache version 3.2.4 for Redis OSS (enhanced)

Amazon ElastiCache version 3.2.4 introduces the next major version of ElastiCache for the Redis OSS engine. ElastiCache 3.2.4 users have all the functionality of earlier Redis OSS versions available to them, plus the option to run in *cluster mode* or *non-cluster mode*. The following table summarizes .

#### Comparing Redis OSS 3.2.4 non-cluster mode and cluster mode

| Feature | Non-cluster mode | Cluster mode |
|---|---|---|
| Data partitioning | No | Yes |
| Geospatial indexing | Yes | Yes |

| Feature | Non-cluster mode | Cluster mode |
| --- | --- | --- |
| Change node type | Yes | Yes * |
| Replica scaling | Yes | Yes * |
| Scale out | No | Yes * |
| Database support | Multiple | Single |
| Parameter group | `default.redis3.2` ** | `default.redis3.2.cluster.on` ** |
| * See Restoring from a backup into a new cache | | |
| ** Or one derived from it. | | |

**Notes:**

- **Partitioning** – the ability to split your data across 2 to 500 node groups (shards) with replication support for each node group.
- **Geospatial indexing** – Redis OSS 3.2.4 introduces support for geospatial indexing via six GEO commands. For more information, see the Redis OSS GEO* command documentation Commands: GEO on the Valkey Commands page (filtered for GEO).

For information about additional Redis OSS 3 features, see Redis OSS 3.2 release notes and Redis OSS 3.0 release notes.

Currently ElastiCache managed Valkey or Redis OSS (cluster mode enabled) does not support the following Redis OSS 3.2 features:

- Replica migration
- Cluster rebalancing
- Lua debugger

ElastiCache disables the following Redis OSS 3.2 management commands:

- `cluster meet`
- `cluster replicate`
- `cluster flushslots`
- `cluster addslots`
- `cluster delslots`
- `cluster setslot`
- `cluster saveconfig`
- `cluster forget`
- `cluster failover`
- `cluster bumpepoch`
- `cluster set-config-epoch`
- `cluster reset`

For information about Redis OSS 3.2.4 parameters, see Redis OSS 3.2.4 parameter changes.

## Past End of Life (EOL) versions (2.x)

### ElastiCache version 2.8.24 for Redis OSS (enhanced)

Redis OSS improvements added since version 2.8.23 include bug fixes and logging of bad memory access addresses. For more information, see Redis OSS 2.8 release notes.

### ElastiCache version 2.8.23 for Redis OSS (enhanced)

Redis OSS improvements added since version 2.8.22 include bug fixes. For more information, see Redis OSS 2.8 release notes. This release also includes support for the new parameter `close-on-slave-write` which, if enabled, disconnects clients who attempt to write to a read-only replica.

For more information on Redis OSS 2.8.23 parameters, see Redis OSS 2.8.23 (enhanced) added parameters in the ElastiCache User Guide.

### ElastiCache version 2.8.22 for Redis OSS (enhanced)

Redis OSS improvements added since version 2.8.21 include the following:

- Support for forkless backups and synchronizations, which allows you to allocate less memory for backup overhead and more for your application. The forkless process can impact both

latency and throughput. When there is high write throughput, when a replica re-syncs, it can be unreachable for the entire time it is syncing.

- If there is a failover, replication groups now recover faster because replicas perform partial syncs with the primary rather than full syncs whenever possible. Additionally, both the primary and replicas no longer use the disk during syncs, providing further speed gains.
- Support for two new CloudWatch metrics.
  - `ReplicationBytes` – The number of bytes a replication group's primary cluster is sending to the read replicas.
  - `SaveInProgress` – A binary value that indicates whether or not there is a background save process running.

  For more information, see Monitoring use with CloudWatch Metrics.
- A number of critical bug fixes in replication PSYNC behavior. For more information, see Redis OSS 2.8 release notes.
- To maintain enhanced replication performance in Multi-AZ replication groups and for increased cluster stability, non-ElastiCache replicas are no longer supported.
- To improve data consistency between the primary cluster and replicas in a replication group, the replicas no longer evict keys independent of the primary cluster.
- Redis OSS configuration variables `appendonly` and `appendfsync` are not supported on Redis OSS version 2.8.22 and later.
- In low-memory situations, clients with a large output buffer might be disconnected from a replica cluster. If disconnected, the client needs to reconnect. Such situations are most likely to occur for PUBSUB clients.

#### ElastiCache version 2.8.21 for Redis OSS

Redis OSS improvements added since version 2.8.19 include a number of bug fixes. For more information, see Redis OSS 2.8 release notes.

#### ElastiCache version 2.8.19 for Redis OSS

Redis OSS improvements added since version 2.8.6 include the following:

- Support for HyperLogLog. For more information, see Redis OSS new data structure: HyperLogLog.

- The sorted set data type now has support for lexicographic range queries with the new commands ZRANGEBYLEX, ZLEXCOUNT, and ZREMRANGEBYLEX.
- To prevent a primary node from sending stale data to replica nodes, the master SYNC fails if a background save (bgsave) child process is aborted.
- Support for the *HyperLogLogBasedCommands* CloudWatch metric. For more information, see Metrics for Valkey and Redis OSS.

#### ElastiCache version 2.8.6 for Redis OSS

Redis OSS improvements added since version 2.6.13 include the following:

- Improved resiliency and fault tolerance for read replicas.
- Support for partial resynchronization.
- Support for user-defined minimum number of read replicas that must be available at all times.
- Full support for pub/sub—notifying clients of events on the server.
- Automatic detection of a primary node failure and failover of your primary node to a secondary node.

#### ElastiCache version 2.6.13 for Redis OSS

ElastiCache version 2.6.13 for Redis OSS was the initial version of ElastiCache that supported Redis OSS. Multi-AZ is not supported on ElastiCache version 2.6.13 for Redis OSS.

## ElastiCache versions for Redis OSS end of life schedule

This section defines end of life (EOL) dates for older major versions as they are announced. This allows you to make version and upgrade decisions for the future.

> **Note**
>
> ElastiCache versions from 5.0.0 to 5.0.5 for Redis OSS are deprecated. Use versions 5.0.6 or greater.

The following table shows the schedule of Extended Support for ElastiCache for Redis OSS engines.

#### Extended Support and End of Life schedule

| Major Engine Version | End of Standard Support | Start of Extended Support Y1 Premium | Start of Extended Support Y2 Premium | Start of Extended Support Y3 Premium | End of Extended Support and version EOL |
|---|---|---|---|---|---|
| Redis OSS v4 | 1/31/2026 | 2/1/2026 | 2/1/2027 | 2/1/2028 | 1/31/2029 |
| Redis OSS v5 | 1/31/2026 | 2/1/2026 | 2/1/2027 | 2/1/2028 | 1/31/2029 |
| Redis OSS v6 | 1/31/2027 | 2/1/2027 | 2/1/2028 | 2/1/2029 | 1/31/2030 |

The following table summarizes each version and its announced EOL date, as well as the recommended upgrade target version.

**Past EOL**

| S M V | Source Minor Versions | Recommended Upgrade Target | EOL Date |
|---|---|---|---|
| V 3 | 3.2.4, 3.2.6 and 3.2.10 | Version 6.2 or higher<br>**Note**<br>For US-ISO-EAST-1, US-ISO-WEST-1, and US-ISOB-E AST-1 Regions, we recommend 5.0.6 or higher. | July 31, 2023 |
| V 2 | 2.8.24, 2.8.23, 2.8.22, 2.8.21, 2.8.19, 2.8.12, 2.8.6, 2.6.13 | Version 6.2 or higher<br>**Note**<br>For US-ISO-EAST-1, US-ISO-WEST-1, | January 13, 2023 |

| S M V | Source Minor Versions | Recommended Upgrade Target | EOL Date |
|---|---|---|---|
| | | and US-ISOB-E AST-1 Regions, we recommend 5.0.6 or higher. | |

## Supported ElastiCache for Memcached versions

ElastiCache supports the following Memcached versions and upgrading to newer versions. When upgrading to a newer version, pay careful attention to the conditions that if not met cause your upgrade to fail.

**ElastiCache for Memcached Versions**

- ElastiCache version 1.6.22 for Memcached
- ElastiCache version 1.6.17 for Memcached
- ElastiCache version 1.6.12 for Memcached
- ElastiCache version 1.6.6 for Memcached
- ElastiCache version 1.5.16 for Memcached
- ElastiCache version 1.5.10 for Memcached
- ElastiCache version 1.4.34 for Memcached
- ElastiCache version 1.4.33 for Memcached
- ElastiCache version 1.4.24 for Memcached
- ElastiCache version 1.4.14 for Memcached
- ElastiCache version 1.4.5 for Memcached

### ElastiCache version 1.6.22 for Memcached

ElastiCache for Memcached version 1.6.22 for Memcached adds support for Memcached version 1.6.22. It includes no new features, but does include bug fixes and cumulative updates from Memcached 1.6.18.

For more information, see ReleaseNotes1622 at Memcached on GitHub.

## ElastiCache version 1.6.17 for Memcached

ElastiCache for Memcached version 1.6.17 for Memcached adds support for Memcached engine version 1.6.17. It includes no new features, but does include bug fixes and cumulative updates from Memcached 1.6.17.

For more information, see ReleaseNotes1617 at Memcached on GitHub.

## ElastiCache version 1.6.12 for Memcached

ElastiCache for Memcached version 1.6.12 for Memcached adds support for Memcached engine 1.6.12 and encryption in-transit. It also includes bug fixes and cumulative updates from Memcached 1.6.6.

For more information, see ReleaseNotes1612 at Memcached on GitHub.

## ElastiCache version 1.6.6 for Memcached

ElastiCache for Memcached version 1.6.6 for Memcached adds support for Memcached version 1.6.6. It includes no new features, but does include bug fixes and cumulative updates from Memcached 1.5.16. ElastiCache for Memcached does not include support for Extstore.

For more information, see ReleaseNotes166 at Memcached on GitHub.

## ElastiCache version 1.5.16 for Memcached

ElastiCache version 1.5.16 for Memcached adds support for Memcached version 1.5.16. It includes no new features, but does include bug fixes and cumulative updates from Memcached 1.5.14 and Memcached 1.5.15.

For more information, see Memcached 1.5.16 Release Notes at Memcached on GitHub.

## ElastiCache version 1.5.10 for Memcached

ElastiCache version 1.5.10 for Memcached supports the following Memcached features:

- Automated slab rebalancing.
- Faster hash table lookups with `murmur3` algorithm.

- Segmented LRU algorithm.
- LRU crawler to background-reclaim memory.
- `--enable-seccomp`: A compile-time option.

It also introduces the `no_modern` and `inline_ascii_resp` parameters. For more information, see Memcached 1.5.10 parameter changes.

Memcached improvements added since ElastiCache version 1.4.34 for Memcached include the following:

- Cumulative fixes, such as ASCII multigets, CVE-2017-9951 and limit crawls for `metadumper`.
- Better connection management by closing connections at the connection limit.
- Improved item-size management for item size above 1MB.
- Better performance and memory-overhead improvements by reducing memory requirements per-item by a few bytes.

For more information, see Memcached 1.5.10 Release Notes at Memcached on GitHub.

### ElastiCache version 1.4.34 for Memcached

ElastiCache version 1.4.34 for Memcached adds no new features to version 1.4.33. Version 1.4.34 is a bug fix release that is larger than the usual such release.

For more information, see Memcached 1.4.34 Release Notes at Memcached on GitHub.

### ElastiCache version 1.4.33 for Memcached

Improvements added since version 1.4.24 include the following:

- Ability to dump all of the metadata for a particular slab class, a list of slab classes, or all slab classes. For more information, see Memcached 1.4.31 Release Notes.
- Improved support for large items over the 1 megabyte default. For more information, see Memcached 1.4.29 Release Notes.
- Ability to specify how long a client can be idle before being asked to close.

  Ability to dynamically increase the amount of memory available to Memcached without having to restart the cluster. For more information, see Memcached 1.4.27 Release Notes.

- Logging of `fetchers`, `mutations`, and `evictions` are now supported. For more information, see Memcached 1.4.26 Release Notes.
- Freed memory can be reclaimed back into a global pool and reassigned to new slab classes. For more information, see Memcached 1.4.25 Release Notes.
- Several bug fixes.
- Some new commands and parameters. For a list, see Memcached 1.4.33 added parameters.

## ElastiCache version 1.4.24 for Memcached

Improvements added since version 1.4.14 include the following:

- Least recently used (LRU) management using a background process.
- Added the option of using *jenkins* or *murmur3* as your hash algorithm.
- Some new commands and parameters. For a list, see Memcached 1.4.24 added parameters.
- Several bug fixes.

## ElastiCache version 1.4.14 for Memcached

Improvements added since version 1.4.5 include the following:

- Enhanced slab rebalancing capability.
- Performance and scalability improvement.
- Introduced the *touch* command to update the expiration time of an existing item without fetching it.
- Auto discovery—the ability for client programs to automatically determine all of the cache nodes in a cluster, and to initiate and maintain connections to all of these nodes.

## ElastiCache version 1.4.5 for Memcached

ElastiCache version 1.4.5 for Memcached was the initial engine and version supported by Amazon ElastiCache for Memcached.

# Major version behavior and compatibility differences for Valkey

When upgrading to a new major Valkey version, review the following behavior changes that may affect your applications.

## Valkey 9.1

Valkey 9.1 introduces the following potentially breaking behavior change compared to Valkey 9.0:

- `CLUSTER SHARDS` and `CLUSTER SLOTS` responses include a new availability-zone field per node. Clients that strictly parse these responses (like older go-redis versions) might reject the unexpected field. Most clients handle unknown fields gracefully. A fix has been provided for go-redis.

For more information on the Valkey 9.1 release, see the Valkey 9.1 release notes.

## Valkey 9.0

Valkey 9.0 introduces the following potentially breaking behavior changes compared to Valkey 8.x:

- Authentication checks run before command validation. Unauthenticated clients now get auth errors instead of "unknown command." (#1475)
- Error messages in transactions include full command name (e.g., `CLIENT SETNAME` instead of `client`). (#2286)
- `GEOSEARCH` shape error message now says "BYRADIUS, BYBOX and BYPOLYGON." (#1809)

For more information on the Valkey 9.0 release, see the Valkey 9.0 release notes.

## Valkey 8.1

Valkey 8.1 introduces zero potentially breaking behavior changes compared to Valkey 8.0.

For more information on the Valkey 8.1 release, see the Valkey 8.1 release notes.

## Valkey 8.0

Valkey 8.0 introduces the following potentially breaking behavior changes compared to Valkey 7.2:

- Nested `MULTI` or `WATCH` inside a transaction now aborts the transaction. Previously silently ignored. (#723)

- SCAN no longer returns lazily expired keys. (#501)
- BITCOUNT and BITPOS return errors instead of zero for invalid arguments on non-existing keys. (Redis#11734)
- Default repl-backlog-size increased 1 MB → 10 MB. (#911)
- Streams use 8 extra bytes per entry. (#688)
- Error messages no longer include "Redis" branding. (#206)

For more information on the Valkey 8.0 release, see the Valkey 8.0 release notes.

### Valkey 7.2

Valkey 7.2.6 has similar compatibility differences with previous versions of Redis OSS 7.2.4. Here are the potentially breaking behavior changes between Valkey 7.2 and Redis OSS 7.1 (or 7.0):

- Freeze time sampling occurs during command execution and in scripts.
- A blocked stream command that's released when key no longer exists carries a different error code (-NOGROUP or -WRONGTYPE instead of -UNBLOCKED).
- Client side tracking for scripts now tracks the keys that are read by the script, instead of the keys that are declared by the caller of EVAL / FCALL.

For more information on the Valkey 7.2 release, see Redis OSS 7.2.4 Release Notes (Valkey 7.2 includes all changes from Redis OSS up to version 7.2.4) and Valkey 7.2 release notes at Valkey on GitHub.

## Major engine version behavior and compatibility differences with Redis OSS

**Important**

The following page is structured to signify all incompability differences between versions and inform you of any considerations you should make when upgrading to newer versions. This list is inclusive of any version incompability issues you may encounter when upgrading. You can upgrade directly from your current Redis OSS version to the latest Redis OSS version available, without the need for sequential upgrades. For example, you can upgrade directly from Redis OSS version 3.0 to version 7.0.

Redis OSS versions are identified with a semantic version which comprise a major, minor, and patch component. For example, in Redis OSS 4.0.10, the major version is 4, the minor version 0, and the patch version is 10. These values are generally incremented based off the following conventions:

- Major versions are for API incompatible changes
- Minor versions are for new functionality added in a backwards-compatible way
- Patch versions are for backwards-compatible bug fixes and non-functional changes

We recommend always staying on the latest patch version within a given **major.minor** version in order to have the latest performance and stability improvements. Beginning with ElastiCache version 6.0 for Redis OSS, ElastiCache will offer a single version for each Redis OSS minor release rather than offering multiple patch versions. ElastiCache will automatically manage the patch version of your running clusters, ensuring improved performance and enhanced security.

We also recommend periodically upgrading to the latest major version, since most major improvements are not back ported to older versions. As ElastiCache expands availability to a new AWS region, ElastiCache for Redis OSS supports the two most recent **major.minor** versions at that time for the new region. For example, if a new AWS region launches and the latest major.minor ElastiCache versions for Redis OSS are **7.0** and **6.2**, ElastiCache will support Redis OSS versions **7.0** and **6.2** in the new AWS region. As newer major.minor versions of ElastiCache for Redis OSS are released, ElastiCache will continue to add support for the newly released versions. To learn more about choosing regions for ElastiCache, see Choosing regions and availability zones.

When doing an upgrade that spans major or minor versions, please consider the following list which includes behavior and backwards incompatible changes released with Redis OSS over time.

## Redis OSS 7.0 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 7.0 release notes.

- `SCRIPT LOAD` and `SCRIPT FLUSH` are no longer propagated to replicas. If you need to have some durability for scripts, we recommend you consider using Redis OSS functions.
- Pubsub channels are now blocked by default for new ACL users.
- `STRALGO` command was replaced with the `LCS` command.
- The format for `ACL GETUSER` has changed so that all fields show the standard access string pattern. If you had automation using `ACL GETUSER`, you should verify that it will handle either format.

- The ACL categories for `SELECT`, `WAIT`, `ROLE`, `LASTSAVE`, `READONLY`, `READWRITE`, and `ASKING` have changed.
- The `INFO` command now shows command stats per sub-command instead of in the top level container commands.
- The return values of `LPOP`, `RPOP`, `ZPOPMIN` and `ZPOPMAX` commands have changed under certain edge cases. If you use these commands, you should check the release notes and evaluate if you are impacted.
- The `SORT` and `SORT_RO` commands now require access to the entire keyspace in order to use the `GET` and `BY` arguments.

## Redis OSS 6.2 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 6.2 release notes.

- The ACL flags of the `TIME`, `ECHO`, `ROLE`, and `LASTSAVE` commands were changed. This may cause commands that were previously allowed to be rejected and vice versa.

> **Note**
>
> None of these commands modify or give access to data.

- When upgrading from Redis OSS 6.0, the ordering of key/value pairs returned from a map response to a lua script are changed. If your scripts use `redis.setresp()` or return a map (new in Redis OSS 6.0), consider the implications that the script may break on upgrades.

## Redis OSS 6.0 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 6.0 release notes.

- The maximum number of allowed databases has been decreased from 1.2 million to 10 thousand. The default value is 16, and we discourage using values much larger than this as we've found performance and memory concerns.
- Set `AutoMinorVersionUpgrade` parameter to yes, and ElastiCache will manage the minor version upgrade through self-service updates. This will be handled through standard customer-notification channels via a self-service update campaign. For more information, see Self-service updates in ElastiCache.

## Redis OSS 5.0 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 5.0 release notes.

- Scripts are by replicated by effects instead of re-executing the script on the replica. This generally improves performance but may increase the amount of data replicated between primaries and replicas. There is an option to revert back to the previous behavior that is only available in ElastiCache version 5.0 for Redis OSS.
- If you are upgrading from Redis OSS 4.0, some commands in LUA scripts will return arguments in a different order than they did in earlier versions. In Redis OSS 4.0, Redis OSS would order some responses lexographically in order to make the responses deterministic, this ordering is not applied when scripts are replicated by effects.
- In Redis OSS 5.0.3 and above, ElastiCache for Redis OSS will offload some IO work to background cores on instance types with more than 4 VCPUs. This may change the performance characteristics Redis OSS and change the values of some metrics. For more information, see Which Metrics Should I Monitor? to understand if you need to change which metrics you watch.

## Redis OSS 4.0 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 4.0 release notes.

- Slow log now logs an additional two arguments, the client name and address. This change should be backwards compatible unless you are explicitly relying on each slow log entry containing 3 values.
- The `CLUSTER NODES` command now returns a slight different format, which is not backwards compatible. We recommend that clients don't use this command for learning about the nodes present in a cluster, and instead they should use `CLUSTER SLOTS`.

## Past EOL

### Redis OSS 3.2 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 3.2 release notes.

- There are no compatibility changes to call out for this version.

For more information, see ElastiCache versions for Redis OSS end of life schedule.

### Redis OSS 2.8 behavior and backwards incompatible changes

For a full list of changes, see Redis OSS 2.8 release notes.

- Starting in Redis OSS 2.8.22, Redis OSS AOF is no longer supported in ElastiCache for Redis OSS. We recommend using MemoryDB when data needs to be persisted durably.
- Starting in Redis OSS 2.8.22, ElastiCache for Redis OSS no longer supports attaching replicas to primaries hosted within ElastiCache. While upgrading, external replicas will be disconnected and they will be unable to reconnect. We recommend using client-side caching, made available in Redis OSS 6.0 as an alternative to external replicas.
- The `TTL` and `PTTL` commands now return -2 if the key does not exist and -1 if it exists but has no associated expire. Redis OSS 2.6 and previous versions used to return -1 for both the conditions.
- `SORT` with `ALPHA` now sorts according to local collation locale if no `STORE` option is used.

For more information, see ElastiCache versions for Redis OSS end of life schedule.

## Upgrade considerations when working with node-based clusters

> **Note**
>
> The following considerations only apply when upgrading node-based clusters. They do not apply to ElastiCache Serverless.

### Valkey and Redis OSS considerations

When upgrading a node-based Valkey or Redis OSS cluster, consider the following.

- Engine version management is designed so that you can have as much control as possible over how patching occurs. However, ElastiCache reserves the right to patch your cluster on your behalf in the unlikely event of a critical security vulnerability in the system or cache software.
- Beginning with ElastiCache version 7.2 for Valkey and ElastiCache version 6.0 for Redis OSS, ElastiCache will offer a single version for each minor release, rather than offering multiple patch versions.
- Starting with Redis OSS engine version 5.0.6, you can upgrade your cluster version with minimal downtime. The cluster is available for reads during the entire upgrade and is available for writes for most of the upgrade duration, except during the failover operation which lasts a few seconds.

- You can also upgrade your ElastiCache clusters with versions earlier than 5.0.6. The process involved is the same but may incur longer failover time during DNS propagation (30s-1m).
- Beginning with Redis OSS 7, ElastiCache supports switching between Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled).
- The Amazon ElastiCache for Redis OSS engine upgrade process is designed to make a best effort to retain your existing data and requires successful Redis OSS replication.
- When upgrading the engine, ElastiCache will terminate existing client connections. To minimize downtime during engine upgrades, we recommend you implement best practices for Redis OSS clients with error retries and exponential backoff and the best practices for minimizing downtime during maintenance.
- You can't upgrade directly from Valkey or Redis OSS (cluster mode disabled) to Valkey or Redis OSS (cluster mode enabled) when you upgrade your engine. The following procedure shows you how to upgrade from Valkey or Redis OSS (cluster mode disabled) to Valkey or Redis OSS (cluster mode enabled).

  **To upgrade from a Valkey or Redis OSS (cluster mode disabled) to Valkey or Redis OSS (cluster mode enabled) engine version**

  1. Make a backup of your Valkey or Redis OSS (cluster mode disabled) cluster or replication group. For more information, see Taking manual backups.
  2. Use the backup to create and seed a Valkey or Redis OSS (cluster mode enabled) cluster with one shard (node group). Specify the new engine version and enable cluster mode when creating the cluster or replication group. For more information, see Tutorial: Seeding a new node-based cluster with an externally created backup.
  3. Delete the old Valkey or Redis OSS (cluster mode disabled) cluster or replication group. For more information, see Deleting a cluster in ElastiCache or Deleting a replication group.
  4. Scale the new Valkey or Redis OSS (cluster mode enabled) cluster or replication group to the number of shards (node groups) that you need. For more information, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters
- When upgrading major engine versions, for example from 5.0.6 to 6.0, you need to also choose a new parameter group that is compatible with the new engine version.
- For single Redis OSS clusters and clusters with Multi-AZ disabled, we recommend that sufficient memory be made available to Redis OSS as described in Ensuring you have enough memory to make a Valkey or Redis OSS snapshot. In these cases, the primary is unavailable to service requests during the upgrade process.

- For Redis OSS clusters with Multi-AZ enabled, we also recommend that you schedule engine upgrades during periods of low incoming write traffic. When upgrading to Redis OSS 5.0.6 or above, the primary cluster continues to be available to service requests during the upgrade process.

  Clusters and replication groups with multiple shards are processed and patched as follows:

  - All shards are processed in parallel. Only one upgrade operation is performed on a shard at any time.
  - In each shard, Amazon ElastiCache creates a new set of nodes running the new engine version. A new node syncs with the existing primary node. After the sync is complete, a failover promotes the new node to primary. The remaining new nodes then sync with the new primary, and the old nodes are removed from the cluster.
  - During this process, there is a brief period where both old and new nodes are visible in the cluster topology. Clients connecting to new replica nodes that are still loading data might receive errors. To handle this transient state, we recommend implementing the section called "Best practices for clients (Valkey and Redis OSS)" with error retries and exponential backoff.
  - Across all the shards, primary failovers are processed in series. Only one primary node is failed over at a time.
- If encryption is enabled on your current cluster or replication group, you cannot upgrade to an engine version that does not support encryption.

**Memcached considerations**

When upgrading a node-based Memcached cluster, consider the following.

- Engine version management is designed so that you can have as much control as possible over how patching occurs. However, ElastiCache reserves the right to patch your cluster on your behalf in the unlikely event of a critical security vulnerability in the system or cache software.
- Because the Memcached engine does not support persistence, Memcached engine version upgrades are always a disruptive process that clears all cache data in the cluster.

# ElastiCache best practices and caching strategies

Below you can find recommended best practices for Amazon ElastiCache. Following these improves your cache's performance and reliability.

**Topics**


- Overall best practices
- Best Practices for using Read Replicas
- Supported and restricted Valkey, Memcached, and Redis OSS commands
- Valkey and Redis OSS configuration and limits
- IPv6 client examples for Valkey, Memcached, and Redis OSS
- Best practices for clients (Valkey and Redis OSS)
- Best practices for clients (Memcached)
- TLS enabled dual stack ElastiCache clusters
- Managing reserved memory for Valkey and Redis OSS
- Best practices when working with Valkey and Redis OSS node-based clusters
- Caching database query results
- Caching strategies for Memcached
- Amazon ElastiCache (Valkey) for e-commerce applications


## Overall best practices

Below you can find information about best practices for using the Valkey, Memcached, and Redis OSS interfaces within ElastiCache.

- **Use cluster-mode enabled configurations** – Cluster-mode enabled allows the cache to scale horizontally to achieve higher storage and throughput than a cluster-mode disabled configuration. ElastiCache serverless is only available in a cluster-mode enabled configuration.
- **Use long-lived connections** – Creating a new connection is expensive, and takes time and CPU resources from the cache. Reuse connections when possible (e.g. with connection pooling) to amortize this cost over many commands.
- **Read from replicas** – If you are using ElastiCache serverless or have provisioned read replicas (node-based clusters), direct reads to replicas to achieve better scalability and/or lower latency. Reads from replicas are eventually consistent with the primary.

  In a node-based cluster, avoid directing read requests to a single read replica since reads may not be available temporarily if the node fails. Either configure your client to direct read requests to at least two read replicas, or direct reads to a single replica and the primary.

In ElastiCache serverless, reading from the replica port (6380) will direct reads to the client's local availability zone when possible, reducing retrieval latency. It will automatically fall back to the other nodes during failures.

- **Avoid expensive commands** – Avoid running any computationally and I/O intensive operations, such as the `KEYS` and `SMEMBERS` commands. We suggest this approach because these operations increase the load on the cluster and have an impact on the performance of the cluster. Instead, use the `SCAN` and `SSCAN` commands.
- **Follow Lua best practices** – Avoid long running Lua scripts, and always declare keys used in Lua scripts up front. We recommend this approach to determine that the Lua script is not using cross slot commands. Ensure that the keys used in Lua scripts belong to the same slot.
- **Use sharded pub/sub** – When using Valkey or Redis OSS to support pub/sub workloads with high throughput, we recommend you use sharded pub/sub (available with Valkey, and with Redis OSS 7 or later). Traditional pub/sub in cluster-mode enabled clusters broadcasts messages to all nodes in the cluster, which can result in high `EngineCPUUtilization`. Note that in ElastiCache serverless, traditional pub/sub commands internally use sharded pub/sub commands.

## Best Practices for using Read Replicas

Many applications, such as session stores, leaderboards, and recommendation engines, require high availability and handle significantly more read operations than write operations. These applications can often tolerate slightly stale data (eventual consistency), meaning that it's acceptable if different users momentarily see slightly different versions of the same data. For example:

- Cached query results can often tolerate slightly stale data, especially for cache-aside patterns where the source of truth is external.
- In a gaming leaderboard, a few seconds delay in updated scores often won't significantly impact the user experience.
- For session stores, some slight delays in propagating session data across replicas rarely affect application functionality.
- Recommendation engines typically use historical data analysis, so real-time consistency is less critical.

Eventual consistency means that all replica nodes will eventually return the same data once the replication process is complete, typically within milliseconds. For such use cases, implementing

read replicas is an effective strategy to reduce latency when reading from your ElastiCache instance.

Using read replicas in Amazon ElastiCache can provide significant performance benefits through:

**Enhanced Read Scalability**

- Distributes read operations across multiple replica nodes
- Offloads read traffic from the primary node
- Reduces read latency by serving requests from geographically closer replicas

**Optimized Primary Node Performance**

- Dedicates primary node resources to write operations
- Reduces connection overhead on the primary node
- Improves write performance and maintains better response times during peak traffic periods

## Using Read from Replica in ElastiCache Serverless

ElastiCache serverless provides two different endpoints, for different consistency requirements. The two endpoints use the same DNS name but different ports. In order to use the read-from-replica port, you must authorize access to both ports from your client application by configuring the security groups and network access control lists of your VPC.

**Primary endpoint (Port 6379)**

- Use for operations requiring strong consistency
- Guarantees reading the most up-to-date data
- Best for critical transactions and write operations
- Necessary for write operations
- Example: `test-12345.serverless.use1.cache.amazonaws.com:6379`

**Read-optimized endpoint (Port 6380)**

- Optimized for read operations that can tolerate eventual consistency
- When possible, ElastiCache serverless automatically routes read requests to a replica node in the client's local Availability Zone. This optimization provides lower latency by avoiding the

additional network latency incurred when retrieving data from a node in a different availability zone.

- ElastiCache serverless automatically selects available nodes in other zones if a local node is unavailable
- Example: `test-12345.serverless.use1.cache.amazonaws.com:6380`
- Clients like Glide and Lettuce will automatically detect and route reads to the latency optimized endpoint if you provide the read from replica configuration. If your client doesn't support routing configuration (e.g., valkey-java and older jedis versions), you must define the right port and client configuration to read from replicas.

## Connecting to read replicas in ElastiCache Serverless - Valkey and Glide

The following code snippet shows how you can configure read from replica for ElastiCache Serverless in the Valkey glide library. You don't need to specify port for read from replicas, but you need to configure the routing configuration `ReadFrom.PREFER_REPLICA`.

```
package glide.examples;

import glide.api.GlideClusterClient;
import glide.api.logging.Logger;
import glide.api.models.configuration.GlideClusterClientConfiguration;
import glide.api.models.configuration.NodeAddress;
import glide.api.models.exceptions.ClosingException;
import glide.api.models.exceptions.ConnectionException;
import glide.api.models.exceptions.TimeoutException;
import glide.api.models.configuration.ReadFrom;

import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;

public class ClusterExample {

    public static void main(String[] args) {
        // Set logger configuration
        Logger.setLoggerConfig(Logger.Level.INFO);

        GlideClusterClient client = null;

        try {
            System.out.println("Connecting to Valkey Glide...");
```

```
            // Configure the Glide Client
            GlideClusterClientConfiguration config =
GlideClusterClientConfiguration.builder()
                .address(NodeAddress.builder()
                    .host("your-endpoint")
                    .port(6379)
                    .build())
                .useTLS(true)
                .readFrom(ReadFrom.PREFER_REPLICA)
                .build();

            // Create the GlideClusterClient
            client = GlideClusterClient.createClient(config).get();
            System.out.println("Connected successfully.");

            // Perform SET operation
            CompletableFuture<String> setResponse = client.set("key", "value");
            System.out.println("Set key 'key' to 'value': " + setResponse.get());

            // Perform GET operation
            CompletableFuture<String> getResponse = client.get("key");
            System.out.println("Get response for 'key': " + getResponse.get());

            // Perform PING operation
            CompletableFuture<String> pingResponse = client.ping();
            System.out.println("PING response: " + pingResponse.get());

        } catch (ClosingException | ConnectionException | TimeoutException |
ExecutionException e) {
            System.err.println("An exception occurred: ");
            e.printStackTrace();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        } finally {
            // Close the client connection
            if (client != null) {
                try {
                    client.close();
                    System.out.println("Client connection closed.");
                } catch (ClosingException | ExecutionException e) {
                    System.err.println("Error closing client: " + e.getMessage());
                }
            }
```

```
        }
    }
}
```

# Supported and restricted Valkey, Memcached, and Redis OSS commands

## Supported Valkey and Redis OSS commands

### Supported Valkey and Redis OSS commands

The following Valkey and Redis OSS commands are supported by serverless caches. Serverless caches also support the search commands. In addition to these commands, these Supported Valkey and Redis OSS commands are also supported.

For information on Bloom Filter commands see Bloom filter commands

#### Bitmap Commands

- `BITCOUNT`

  Counts the number of set bits (population counting) in a string.

  Learn more
- `BITFIELD`

  Performs arbitrary bitfield integer operations on strings.

  Learn more
- `BITFIELD_RO`

  Performs arbitrary read-only bitfield integer operations on strings.

  Learn more
- `BITOP`

  Performs bitwise operations on multiple strings, and stores the result.

  Learn more
- `BITPOS`

Finds the first set (1) or clear (0) bit in a string.

Learn more

- `GETBIT`

  Returns a bit value by offset.

  Learn more

- `SETBIT`

  Sets or clears the bit at offset of the string value. Creates the key if it doesn't exist.

  Learn more

**Cluster Management Commands**

- `CLUSTER COUNTKEYSINSLOT`

  Returns the number of keys in a hash slot.

  Learn more

- `CLUSTER GETKEYSINSLOT`

  Returns the key names in a hash slot.

  Learn more

- `CLUSTER INFO`

  Returns information about the state of a node. In a serverless cache, returns state about the single virtual “shard” exposed to the client.

  Learn more

- `CLUSTER KEYSLOT`

  Returns the hash slot for a key. In Valkey 9.1 and later, available in both cluster mode and standalone mode.

  Learn more

- `CLUSTER MYID`

Returns the ID of a node. In a serverless cache, returns state about the single virtual “shard” exposed to the client.

Learn more

- `CLUSTER NODES`

  Returns the cluster configuration for a node. In a serverless cache, returns state about the single virtual “shard” exposed to the client.

  Learn more

- `CLUSTER REPLICAS`

  Lists the replica nodes of a master node. In a serverless cache, returns state about the single virtual “shard” exposed to the client.

  Learn more

- `CLUSTER SHARDS`

  Returns the mapping of cluster slots to shards. In Valkey 9.1 and later, includes an `availability-zone` field per node. In a serverless cache, returns state about the single virtual “shard” exposed to the client.

  Learn more

- `CLUSTER SLOTS`

  Returns the mapping of cluster slots to nodes. In a serverless cache, returns state about the single virtual “shard” exposed to the client.

  Learn more

- `CLUSTERSCAN`

  Iterates over key names across all nodes in a cluster. Available in cluster-mode-enabled configurations.

  Learn more

- `CLUSTER SLOT-STATS`

  Allows tracking of per slot metrics for key count, CPU utilization, network bytes in, and network bytes out.

Learn more

- READONLY

  Enables read-only queries for a connection to a Valkey or Redis OSS Cluster replica node.

  Learn more

- READWRITE

  Enables read-write queries for a connection to a Valkey or Redis OSS Cluster replica node.

  Learn more

- SCRIPT SHOW

  Returns the original source code of a script in the script cache.

  Learn more

**Connection Management Commands**

- AUTH

  Authenticates the connection.

  Learn more

- CLIENT GETNAME

  Returns the name of the connection.

  Learn more

- CLIENT REPLY

  Instructs the server whether to reply to commands.

  Learn more

- CLIENT SETNAME

  Sets the connection name.

  Learn more

- `ECHO`

  Returns the given string.

  Learn more

- `HELLO`

  Handshakes with the Valkey or Redis OSS server.

  Learn more

- `PING`

  Returns the server's liveliness response.

  Learn more

- `QUIT`

  Closes the connection.

  Learn more

- `RESET`

  Resets the connection.

  Learn more

- `SELECT`

  Changes the selected database.

  Learn more

**Generic Commands**

- `COPY`

  Copies the value of a key to a new key.

  Learn more

- `DEL`

Deletes one or more keys.

Learn more

- DELIFEQ

  Deletes the given key only if its current value is equal to the provided value.

  Learn more

- DUMP

  Returns a serialized representation of the value stored at a key.

  Learn more

- EXISTS

  Determines whether one or more keys exist.

  Learn more

- EXPIRE

  Sets the expiration time of a key in seconds.

  Learn more

- EXPIREAT

  Sets the expiration time of a key to a Unix timestamp.

  Learn more

- EXPIRETIME

  Returns the expiration time of a key as a Unix timestamp.

  Learn more

- PERSIST

  Removes the expiration time of a key.

  Learn more

- PEXPIRE

Sets the expiration time of a key in milliseconds.

Learn more

- PEXPIREAT

  Sets the expiration time of a key to a Unix milliseconds timestamp.

  Learn more

- PEXPIRETIME

  Returns the expiration time of a key as a Unix milliseconds timestamp.

  Learn more

- PTTL

  Returns the expiration time in milliseconds of a key.

  Learn more

- RANDOMKEY

  Returns a random key name from the database.

  Learn more

- RENAME

  Renames a key and overwrites the destination.

  Learn more

- RENAMENX

  Renames a key only when the target key name doesn't exist.

  Learn more

- RESTORE

  Creates a key from the serialized representation of a value.

  Learn more

- SCAN

Iterates over the key names in the database.

Learn more

> **Note**
>
> Frequent or large SCAN operations are resource intensive and can increase CPU utilization, which might raise latency for other operations. On serverless caches, heavy SCAN usage can slow the scaling (resharding) process, which might cause scaling to take longer than usual.
> We recommend that you review whether your workload requires frequent full keyspace scans and reduce them where possible. For serverless caches, consider configuring a minimum capacity to maintain headroom.

- SORT

  Sorts the elements in a list, a set, or a sorted set, optionally storing the result.

  Learn more

- SORT_RO

  Returns the sorted elements of a list, a set, or a sorted set.

  Learn more

- TOUCH

  Returns the number of existing keys out of those specified after updating the time they were last accessed.

  Learn more

- TTL

  Returns the expiration time in seconds of a key.

  Learn more

- TYPE

  Determines the type of value stored at a key.

Learn more

- UNLINK

  Asynchronously deletes one or more keys.

  Learn more

### Geospatial Commands

- GEOADD

  Adds one or more members to a geospatial index. The key is created if it doesn't exist.

  Learn more

- GEODIST

  Returns the distance between two members of a geospatial index.

  Learn more

- GEOHASH

  Returns members from a geospatial index as geohash strings.

  Learn more

- GEOPOS

  Returns the longitude and latitude of members from a geospatial index.

  Learn more

- GEORADIUS

  Queries a geospatial index for members within a distance from a coordinate, optionally stores the result.

  Learn more

- GEORADIUS_RO

  Returns members from a geospatial index that are within a distance from a coordinate.

Learn more

- `GEORADIUSBYMEMBER`

  Queries a geospatial index for members within a distance from a member, optionally stores the result.

  Learn more
- `GEORADIUSBYMEMBER_RO`

  Returns members from a geospatial index that are within a distance from a member.

  Learn more
- `GEOSEARCH`

  Queries a geospatial index for members inside an area of a box or a circle.

  Learn more
- `GEOSEARCHSTORE`

  Queries a geospatial index for members inside an area of a box or a circle, optionally stores the result.

  Learn more

**Hash Commands**

- `HDEL`

  Deletes one or more fields and their values from a hash. Deletes the hash if no fields remain.

  Learn more
- `HEXISTS`

  Determines whether a field exists in a hash.

  Learn more
- `HGET`

  Returns the value of a field in a hash.

Learn more

- HGETALL

  Returns all fields and values in a hash.

  Learn more

- HINCRBY

  Increments the integer value of a field in a hash by a number. Uses 0 as initial value if the field doesn't exist.

  Learn more

- HINCRBYFLOAT

  Increments the floating point value of a field by a number. Uses 0 as initial value if the field doesn't exist.

  Learn more

- HKEYS

  Returns all fields in a hash.

  Learn more

- HLEN

  Returns the number of fields in a hash.

  Learn more

- HMGET

  Returns the values of all fields in a hash.

  Learn more

- HMSET

  Sets the values of multiple fields.

  Learn more

- HRANDFIELD

Returns one or more random fields from a hash.

Learn more

- HSCAN

  Iterates over fields and values of a hash.

  Learn more

- HSET

  Creates or modifies the value of a field in a hash.

  Learn more

- HSETNX

  Sets the value of a field in a hash only when the field doesn't exist.

  Learn more

- HSTRLEN

  Returns the length of the value of a field.

  Learn more

- HVALS

  Returns all values in a hash.

  Learn more

- HSETEX

  Sets fields in a hash with per-field expiration times. Supports NX (set only if field does not exist) and XX (set only if field already exists) flags. Creates the key if it doesn't exist.

  Learn more

- HGETEX

  Returns the values of fields in a hash and optionally sets their expiration.

  Learn more

- HGETDEL

  Returns the values of fields in a hash and deletes them.

  Learn more

- HEXPIRE

  Sets the expiration time of a hash field in seconds.

  Learn more

- HEXPIREAT

  Sets the expiration time of a hash field to a Unix timestamp.

  Learn more

- HPEXPIRE

  Sets the expiration time of a hash field in milliseconds.

  Learn more

- HPEXPIREAT

  Sets the expiration time of a hash field to a Unix milliseconds timestamp.

  Learn more

- HPERSIST

  Removes the expiration time from a hash field.

  Learn more

- HTTL

  Returns the remaining time to live of a hash field in seconds.

  Learn more

- HPTTL

  Returns the remaining time to live of a hash field in milliseconds.

  Learn more

- HEXPIRETIME

  Returns the expiration time of a hash field as a Unix timestamp.

  Learn more

- HPEXPIRETIME

  Returns the expiration time of a hash field as a Unix milliseconds timestamp.

  Learn more

**HyperLogLog Commands**

- PFADD

  Adds elements to a HyperLogLog key. Creates the key if it doesn't exist.

  Learn more

- PFCOUNT

  Returns the approximated cardinality of the set(s) observed by the HyperLogLog key(s).

  Learn more

- PFMERGE

  Merges one or more HyperLogLog values into a single key.

  Learn more

**List Commands**

- BLMOVE

  Pops an element from a list, pushes it to another list and returns it. Blocks until an element is available otherwise. Deletes the list if the last element was moved.

  Learn more

- BLMPOP

Pops the first element from one of multiple lists. Blocks until an element is available otherwise. Deletes the list if the last element was popped.

Learn more

- BLPOP

  Removes and returns the first element in a list. Blocks until an element is available otherwise. Deletes the list if the last element was popped.

  Learn more

- BRPOP

  Removes and returns the last element in a list. Blocks until an element is available otherwise. Deletes the list if the last element was popped.

  Learn more

- BRPOPLPUSH

  Pops an element from a list, pushes it to another list and returns it. Block until an element is available otherwise. Deletes the list if the last element was popped.

  Learn more

- LINDEX

  Returns an element from a list by its index.

  Learn more

- LINSERT

  Inserts an element before or after another element in a list.

  Learn more

- LLEN

  Returns the length of a list.

  Learn more

- LMOVE

Returns an element after popping it from one list and pushing it to another. Deletes the list if the last element was moved.

Learn more

- LMPOP

  Returns multiple elements from a list after removing them. Deletes the list if the last element was popped.

  Learn more

- LPOP

  Returns the first elements in a list after removing it. Deletes the list if the last element was popped.

  Learn more

- LPOS

  Returns the index of matching elements in a list.

  Learn more

- LPUSH

  Prepends one or more elements to a list. Creates the key if it doesn't exist.

  Learn more

- LPUSHX

  Prepends one or more elements to a list only when the list exists.

  Learn more

- LRANGE

  Returns a range of elements from a list.

  Learn more

- LREM

  Removes elements from a list. Deletes the list if the last element was removed.

Learn more

- LSET

  Sets the value of an element in a list by its index.

  Learn more

- LTRIM

  Removes elements from both ends a list. Deletes the list if all elements were trimmed.

  Learn more

- RPOP

  Returns and removes the last elements of a list. Deletes the list if the last element was popped.

  Learn more

- RPOPLPUSH

  Returns the last element of a list after removing and pushing it to another list. Deletes the list if the last element was popped.

  Learn more

- RPUSH

  Appends one or more elements to a list. Creates the key if it doesn't exist.

  Learn more

- RPUSHX

  Appends an element to a list only when the list exists.

  Learn more

### Pub/Sub Commands

> **Note**
>
> PUBSUB commands internally use sharded PUBSUB, so channel names will be mixed.

- PUBLISH

  Posts a message to a channel.

  Learn more

- PUBSUB CHANNELS

  Returns the active channels.

  Learn more

- PUBSUB NUMSUB

  Returns a count of subscribers to channels.

  Learn more

- PUBSUB SHARDCHANNELS

  Returns the active shard channels.

  Learn more

- PUBSUB SHARDNUMSUB

  Returns the count of subscribers of shard channels.

  Learn more

- SPUBLISH

  Post a message to a shard channel

  Learn more

- SSUBSCRIBE

  Listens for messages published to shard channels.

  Learn more

- SUBSCRIBE

  Listens for messages published to channels.

  Learn more

- SUNSUBSCRIBE

  Stops listening to messages posted to shard channels.

  Learn more

- UNSUBSCRIBE

  Stops listening to messages posted to channels.

  Learn more

**Scripting Commands**

- EVAL

  Executes a server-side Lua script.

  Learn more

- EVAL_RO

  Executes a read-only server-side Lua script.

  Learn more

- EVALSHA

  Executes a server-side Lua script by SHA1 digest.

  Learn more

- EVALSHA_RO

  Executes a read-only server-side Lua script by SHA1 digest.

  Learn more

- SCRIPT EXISTS

  Determines whether server-side Lua scripts exist in the script cache.

  Learn more

- SCRIPT FLUSH

Currently a no-op, script cache is managed by the service.

Learn more

- `SCRIPT LOAD`

  Loads a server-side Lua script to the script cache.

  Learn more

### Server Management Commands

> **Note**
>
> When using node-based ElastiCache clusters for Valkey and Redis OSS, flush commands must be sent to every primary by the client to flush all keys. ElastiCache Serverless for Valkey and Redis OSS works differently, because it abstracts away the underlying cluster topology. The result is that in ElastiCache Serverless, `FLUSHDB` and `FLUSHALL` commands will always flush all keys across the cluster. For this reason, flush commands cannot be included inside a Serverless transaction.

- `ACL CAT`

  Lists the ACL categories, or the commands inside a category.

  Learn more

- `ACL GENPASS`

  Generates a pseudorandom, secure password that can be used to identify ACL users.

  Learn more

- `ACL GETUSER`

  Lists the ACL rules of a user.

  Learn more

- `ACL LIST`

  Dumps the effective rules in ACL file format.

Learn more

- `ACL USERS`

  Lists all ACL users.

  Learn more

- `ACL WHOAMI`

  Returns the authenticated username of the current connection.

  Learn more

- `DBSIZE`

  Return the number of keys in the currently-selected database. This operation is not guaranteed to be atomic across all slots.

  Learn more

- `COMMAND`

  Returns detailed information about all commands.

  Learn more

- `COMMAND COUNT`

  Returns a count of commands.

  Learn more

- `COMMAND DOCS`

  Returns documentary information about one, multiple or all commands.

  Learn more

- `COMMAND GETKEYS`

  Extracts the key names from an arbitrary command.

  Learn more

- `COMMAND GETKEYSANDFLAGS`

Extracts the key names and access flags for an arbitrary command.

Learn more

- `COMMAND INFO`

  Returns information about one, multiple or all commands.

  Learn more

- `COMMAND LIST`

  Returns a list of command names.

  Learn more

- `COMMANDLOG`

  A container for command log commands.

  Learn more

- `COMMANDLOG GET`

  Returns the specified command log's entries.

  Learn more

- `COMMANDLOG HELP`

  Show helpful text about the different subcommands.

  Learn more

- `COMMANDLOG LEN`

  Returns the number of entries in the specified type of command log.

  Learn more

- `COMMANDLOG RESET`

  Clears all entries from the specified type of command log.

  Learn more

- `FLUSHALL`

Removes all keys from all databases. This operation is not guaranteed to be atomic across all slots.

Learn more

- `FLUSHDB`

  Remove all keys from the current database. This operation is not guaranteed to be atomic across all slots.

  Learn more

- `INFO`

  Returns information and statistics about the server.

  Learn more

- `LOLWUT`

  Displays computer art and the Valkey or Redis OSS version.

  Learn more

- `ROLE`

  Returns the replication role.

  Learn more

- `TIME`

  Returns the server time.

  Learn more

**Set Commands**

- `SADD`

  Adds one or more members to a set. Creates the key if it doesn't exist.

  Learn more

- `SCARD`

Returns the number of members in a set.

Learn more

- SDIFF

  Returns the difference of multiple sets.

  Learn more

- SDIFFSTORE

  Stores the difference of multiple sets in a key.

  Learn more

- SINTER

  Returns the intersect of multiple sets.

  Learn more

- SINTERCARD

  Returns the number of members of the intersect of multiple sets.

  Learn more

- SINTERSTORE

  Stores the intersect of multiple sets in a key.

  Learn more

- SISMEMBER

  Determines whether a member belongs to a set.

  Learn more

- SMEMBERS

  Returns all members of a set.

  Learn more

- SMISMEMBER

Determines whether multiple members belong to a set.

Learn more

- SMOVE

  Moves a member from one set to another.

  Learn more

- SPOP

  Returns one or more random members from a set after removing them. Deletes the set if the last member was popped.

  Learn more

- SRANDMEMBER

  Get one or multiple random members from a set

  Learn more

- SREM

  Removes one or more members from a set. Deletes the set if the last member was removed.

  Learn more

- SSCAN

  Iterates over members of a set.

  Learn more

- SUNION

  Returns the union of multiple sets.

  Learn more

- SUNIONSTORE

  Stores the union of multiple sets in a key.

  Learn more

### Sorted Set Commands

- BZMPOP

  Removes and returns a member by score from one or more sorted sets. Blocks until a member is available otherwise. Deletes the sorted set if the last element was popped.

  Learn more

- BZPOPMAX

  Removes and returns the member with the highest score from one or more sorted sets. Blocks until a member available otherwise. Deletes the sorted set if the last element was popped.

  Learn more

- BZPOPMIN

  Removes and returns the member with the lowest score from one or more sorted sets. Blocks until a member is available otherwise. Deletes the sorted set if the last element was popped.

  Learn more

- ZADD

  Adds one or more members to a sorted set, or updates their scores. Creates the key if it doesn't exist.

  Learn more

- ZCARD

  Returns the number of members in a sorted set.

  Learn more

- ZCOUNT

  Returns the count of members in a sorted set that have scores within a range.

  Learn more

- ZDIFF

  Returns the difference between multiple sorted sets.

Learn more

- `ZDIFFSTORE`

  Stores the difference of multiple sorted sets in a key.

  Learn more

- `ZINCRBY`

  Increments the score of a member in a sorted set.

  Learn more

- `ZINTER`

  Returns the intersect of multiple sorted sets.

  Learn more

- `ZINTERCARD`

  Returns the number of members of the intersect of multiple sorted sets.

  Learn more

- `ZINTERSTORE`

  Stores the intersect of multiple sorted sets in a key.

  Learn more

- `ZLEXCOUNT`

  Returns the number of members in a sorted set within a lexicographical range.

  Learn more

- `ZMPOP`

  Returns the highest- or lowest-scoring members from one or more sorted sets after removing them. Deletes the sorted set if the last member was popped.

  Learn more

- `ZMSCORE`

Returns the score of one or more members in a sorted set.

Learn more

- ZPOPMAX

  Returns the highest-scoring members from a sorted set after removing them. Deletes the sorted set if the last member was popped.

  Learn more

- ZPOPMIN

  Returns the lowest-scoring members from a sorted set after removing them. Deletes the sorted set if the last member was popped.

  Learn more

- ZRANDMEMBER

  Returns one or more random members from a sorted set.

  Learn more

- ZRANGE

  Returns members in a sorted set within a range of indexes.

  Learn more

- ZRANGEBYLEX

  Returns members in a sorted set within a lexicographical range.

  Learn more

- ZRANGEBYSCORE

  Returns members in a sorted set within a range of scores.

  Learn more

- ZRANGESTORE

  Stores a range of members from sorted set in a key.

Learn more

- ZRANK

  Returns the index of a member in a sorted set ordered by ascending scores.

  Learn more

- ZREM

  Removes one or more members from a sorted set. Deletes the sorted set if all members were removed.

  Learn more

- ZREMRANGEBYLEX

  Removes members in a sorted set within a lexicographical range. Deletes the sorted set if all members were removed.

  Learn more

- ZREMRANGEBYRANK

  Removes members in a sorted set within a range of indexes. Deletes the sorted set if all members were removed.

  Learn more

- ZREMRANGEBYSCORE

  Removes members in a sorted set within a range of scores. Deletes the sorted set if all members were removed.

  Learn more

- ZREVRANGE

  Returns members in a sorted set within a range of indexes in reverse order.

  Learn more

- ZREVRANGEBYLEX

  Returns members in a sorted set within a lexicographical range in reverse order.

Learn more

- ZREVRANGEBYSCORE

  Returns members in a sorted set within a range of scores in reverse order.

  Learn more

- ZREVRANK

  Returns the index of a member in a sorted set ordered by descending scores.

  Learn more

- ZSCAN

  Iterates over members and scores of a sorted set.

  Learn more

- ZSCORE

  Returns the score of a member in a sorted set.

  Learn more

- ZUNION

  Returns the union of multiple sorted sets.

  Learn more

- ZUNIONSTORE

  Stores the union of multiple sorted sets in a key.

  Learn more

**Stream Commands**

- XACK

  Returns the number of messages that were successfully acknowledged by the consumer group member of a stream.

Learn more

- XADD

  Appends a new message to a stream. Creates the key if it doesn't exist.

  Learn more

- XAUTOCLAIM

  Changes, or acquires, ownership of messages in a consumer group, as if the messages were delivered to as consumer group member.

  Learn more

- XCLAIM

  Changes, or acquires, ownership of a message in a consumer group, as if the message was delivered a consumer group member.

  Learn more

- XDEL

  Returns the number of messages after removing them from a stream.

  Learn more

- XGROUP CREATE

  Creates a consumer group.

  Learn more

- XGROUP CREATECONSUMER

  Creates a consumer in a consumer group.

  Learn more

- XGROUP DELCONSUMER

  Deletes a consumer from a consumer group.

  Learn more

- XGROUP DESTROY

Destroys a consumer group.

Learn more

- `XGROUP SETID`

  Sets the last-delivered ID of a consumer group.

  Learn more

- `XINFO CONSUMERS`

  Returns a list of the consumers in a consumer group.

  Learn more

- `XINFO GROUPS`

  Returns a list of the consumer groups of a stream.

  Learn more

- `XINFO STREAM`

  Returns information about a stream.

  Learn more

- `XLEN`

  Return the number of messages in a stream.

  Learn more

- `XPENDING`

  Returns the information and entries from a stream consumer group's pending entries list.

  Learn more

- `XRANGE`

  Returns the messages from a stream within a range of IDs.

  Learn more

- `XREAD`

Returns messages from multiple streams with IDs greater than the ones requested. Blocks until a message is available otherwise.

Learn more

- XREADGROUP

  Returns new or historical messages from a stream for a consumer in a group. Blocks until a message is available otherwise.

  Learn more

- XREVRANGE

  Returns the messages from a stream within a range of IDs in reverse order.

  Learn more

- XTRIM

  Deletes messages from the beginning of a stream.

  Learn more

**String Commands**

- APPEND

  Appends a string to the value of a key. Creates the key if it doesn't exist.

  Learn more

- DECR

  Decrements the integer value of a key by one. Uses 0 as initial value if the key doesn't exist.

  Learn more

- DECRBY

  Decrements a number from the integer value of a key. Uses 0 as initial value if the key doesn't exist.

  Learn more

- `GET`

  Returns the string value of a key.

  Learn more

- `GETDEL`

  Returns the string value of a key after deleting the key.

  Learn more

- `GETEX`

  Returns the string value of a key after setting its expiration time.

  Learn more

- `GETRANGE`

  Returns a substring of the string stored at a key.

  Learn more

- `GETSET`

  Returns the previous string value of a key after setting it to a new value.

  Learn more

- `INCR`

  Increments the integer value of a key by one. Uses 0 as initial value if the key doesn't exist.

  Learn more

- `INCRBY`

  Increments the integer value of a key by a number. Uses 0 as initial value if the key doesn't exist.

  Learn more

- `INCRBYFLOAT`

  Increment the floating point value of a key by a number. Uses 0 as initial value if the key doesn't exist.

Learn more

- LCS

  Finds the longest common substring.

  Learn more

- MGET

  Atomically returns the string values of one or more keys.

  Learn more

- MSET

  Atomically creates or modifies the string values of one or more keys.

  Learn more

- MSETNX

  Atomically modifies the string values of one or more keys only when all keys don't exist.

  Learn more

- MSETEX

  Atomically sets multiple keys with a shared expiration time.

  Learn more

- PSETEX

  Sets both string value and expiration time in milliseconds of a key. The key is created if it doesn't exist.

  Learn more

- SET

  Sets the string value of a key, ignoring its type. The key is created if it doesn't exist.

  Learn more

- SETEX

Sets the string value and expiration time of a key. Creates the key if it doesn't exist.

Learn more

- SETNX

  Set the string value of a key only when the key doesn't exist.

  Learn more

- SETRANGE

  Overwrites a part of a string value with another by an offset. Creates the key if it doesn't exist.

  Learn more

- STRLEN

  Returns the length of a string value.

  Learn more

- SUBSTR

  Returns a substring from a string value.

  Learn more

**Transaction Commands**

- DISCARD

  Discards a transaction.

  Learn more

- EXEC

  Executes all commands in a transaction.

  Learn more

- MULTI

  Starts a transaction.

Learn more

## Search commands

### Search Commands

The following search commands are supported by ElastiCache for Valkey.

- `FT.CREATE`

  Creates an index and initiates a backfill of that index.

  Learn more

- `FT.SEARCH`

  Uses the provided query expression to locate keys within an index.

  Learn more

- `FT.AGGREGATE`

  Runs an aggregation pipeline against an index.

  Learn more

- `FT.DROPINDEX`

  Deletes an index.

  Learn more

- `FT.INFO`

  Returns information about a given index.

  Learn more

- `FT._LIST`

  Returns a list of all existing indexes.

  Learn more

## Restricted Valkey and Redis OSS commands

To deliver a managed service experience, ElastiCache restricts access to certain cache engine-specific commands that require advanced privileges. For caches running Redis OSS, the following commands are unavailable:

- `acl setuser`
- `acl load`
- `acl save`
- `acl deluser`
- `bgrewriteaof`
- `bgsave`
- `cluster addslot`
- `cluster addslotsrange`
- `cluster bumpepoch`
- `cluster delslot`
- `cluster delslotsrange`
- `cluster failover`
- `cluster flushslots`
- `cluster forget`
- `cluster links`
- `cluster meet`
- `cluster setslot`
- `config`
- `debug`
- `migrate`
- `psync`
- `replicaof`
- `save`
- `slaveof`
- `shutdown`

- sync

In addition, the following commands are unavailable for serverless caches:

- acl log
- client caching
- client getredir
- client id
- client info
- client kill
- client list
- client no-evict
- client pause
- client tracking
- client trackinginfo
- client unblock
- client unpause
- cluster count-failure-reports
- commandlog
- commandlog get
- commandlog help
- commandlog len
- commandlog reset
- fcall
- fcall_ro
- function
- function delete
- function dump
- function flush
- function help

- function kill
- function list
- function load
- function restore
- function stats
- keys
- lastsave
- latency
- latency doctor
- latency graph
- latency help
- latency histogram
- latency history
- latency latest
- latency reset
- memory
- memory doctor
- memory help
- memory malloc-stats
- memory purge
- memory stats
- memory usage
- monitor
- move
- object
- object encoding
- object freq
- object help
- object idletime
- object refcount

- `pfdebug`
- `pfselftest`
- `psubscribe`
- `pubsub numpat`
- `punsubscribe`
- `script kill`
- `slowlog`
- `slowlog get`
- `slowlog help`
- `slowlog len`
- `slowlog reset`
- `swapdb`
- `wait`

## Supported Memcached commands

ElastiCache Serverless for Memcached supports all of the memcached commands in open source memcached 1.6 except for the following:

- Client connections require TLS, as a result UDP protocol is not supported.
- Binary protocol is not supported, as it is officially deprecated in memcached 1.6.
- GET/GETS commands are limited to 16KB to avoid potential DoS attack to the server with fetching large number of keys.
- Delayed `flush_all` command will be rejected with `CLIENT_ERROR`.
- Commands that configure the engine or reveal internal information about engine state or logs are not supported, such as:
  - For STATS command, only `stats` and `stats reset` are supported. Other variations will return ERROR
  - `lru / lru_crawler` - modification for LRU and LRU crawler settings
  - `watch` - watches memcached server logs
  - `verbosity` - configures the server log level
  - `me` - meta debug (me) command is not supported

# Valkey and Redis OSS configuration and limits

The Valkey and Redis OSS engines each provides a number of configuration parameters, some of which are modifiable in ElastiCache for Redis OSS and some of which are not modifiable to provide stable performance and reliability.

## Serverless caches

For serverless caches, parameter groups are not used and all Valkey or Redis OSS configuration is not modifiable. The following Valkey or Redis OSS parameters are in place:

| Name | Details | Description |
|---|---|---|
| `acl-pubsub-default` | `allchannels` | Default pubsub channel permissions for ACL users on the cache. |
| `client-output-buffer-limit` | `normal 0 0 0`<br><br>`pubsub 32mb 8mb 60` | Normal clients have no buffer limit. PUB/SUB clients will be disconnected if they breach 32MiB backlog, or breach 8MiB backlog for 60s. |
| `client-query-buffer-limit` | 1 GiB | The maximum size of a single client query buffer. Additionally, clients cannot issue a request with more than 39,999 arguments. |
| `cluster-allow-pubsubshard-when-down` | `yes` | This allows the cache to serve pubsub traffic while the cache is partially down. |
| `cluster-allow-reads-when-down` | `yes` | This allows the cache to serve read traffic while the cache is partially down. |
| `cluster-databases` | `1024` | Starting with version 9.0, Valkey cluster supports multiple databases, similar to standalone mode but with some additional |

| Name | Details | Description |
| --- | --- | --- |
| | | restrictions. Serverless caches support up to 1024 databases. |
| `cluster-enabled` | `yes` | All serverless caches are cluster mode enabled, which allows them to transparently partition their data across multiple backend shards. All slots are surfaced to clients as being owned by a single virtual node. |
| `cluster-require-full-coverage` | no | When the keyspace is partially down (i.e. at least one hash slot is inaccessible), the cache will continue accepting queries for the part of the keyspace that is still covered. The entire keyspace will always be "covered" by a single virtual node in `cluster slots`. |
| `lua-time-limit` | 5000 | The maximum execution time for a Lua script, in milliseconds, before ElastiCache takes action to stop the script.<br><br>If `lua-time-limit` is exceeded, all Valkey or Redis OSS commands may return an error of the form ____*-BUSY*. Since this state can cause interference with many essential Valkey or Redis OSS operations, ElastiCache will first issue a *SCRIPT KILL* command. If this is unsuccessful, ElastiCache will forcibly restart Valkey or Redis OSS. |
| `maxclients` | 65000 | The maximum number of clients that can be connected to the cache at one time. Further connections established may or may not succeed. |

| Name | Details | Description |
| --- | --- | --- |
| `maxmemory-policy` | `volatile-lru` | Items with a TTL set are evicted following least-recently-used (LRU) estimation when a cache's memory limit is reached. |
| `notify-keyspace-events` | (an empty string) | Keyspace events are currently not supported on serverless caches. |
| `port` | Primary port: 6379<br><br>Read port: 6380 | Serverless caches advertise two ports with the same hostname. The primary port allows writes and reads, whereas the read port allows lower-latency eventually-consistent reads using the `READONLY` command. |
| `proto-max-bulk-len` | 512 MiB | The maximum size of a single element request. |
| `timeout` | 0 | Clients are not forcibly disconnected at a specific idle time, but they may be disconnected during steady-state for load balancing purposes. |

Additionally, the following limits are in place:

| Name | Details | Description |
| --- | --- | --- |
| Size per cache | 5,000 GiB | Maximum amount of data that can be stored per serverless cache. |
| Size per slot | 32 GiB | The maximum size of a single Valkey or Redis OSS hash slot. Clients trying to set more data than this on a single Valkey or Redis OSS slot will trigger the eviction policy on the slot, and |

| Name | Details | Description |
| --- | --- | --- |
| | | if no keys are evictable, will receive an out of memory (OOM) error. |
| ECPU per cache | 15,000,000 ECPU/second | ElastiCache Processing Units (ECPU) metric. The number of ECPUs consumed by your requests depends on the vCPU time taken and the amount of data transferred. |
| ECPU per slot | 30K - 90K ECPU/second | Maximum of 30K ECPUs/second per slot or 90K ECPUs/second when using Read from Replica using READONLY connections. |
| Arguments per Request | 39,999 | Maximum number of arguments per request. Clients sending more arguments per request will receive an error. |
| Key name length | 4 KiB | The maximum size for a single Valkey or Redis OSS key or channel name. Clients referencing keys larger than this will receive an error. |
| Lua script size | 4 MiB | The maximum size of a single Valkey or Redis OSS Lua script. Attempts to load a Lua script larger than this will receive an error. |

### Node-based clusters

For node-based clusters, see Valkey and Redis OSS parameters for the default values of configuration parameters and which are configurable. The default values are generally recommended unless you have a specific use case requiring them to be overridden.

## IPv6 client examples for Valkey, Memcached, and Redis OSS

ElastiCache is compatible with Valkey, Memcached, and Redis OSS. This means that clients which support IPv6 connections should be able to connect to IPv6 enabled ElastiCache for Memcached clusters. There are some caveats worth noting when interacting with IPv6 enabled resources.

You can view the Best practices for Valkey and Redis clients blog post on the AWS Database Blog for recommendations on configuring Valkey and Redis OSS clients for ElastiCache resources.

Following are best practices for interacting with IPv6 enabled ElastiCache resources with commonly used open-source client libraries.

### Validated clients with Valkey and Redis OSS

ElastiCache is compatible with Valkey and open-source Redis OSS. This means that Valkey and open source Redis OSS clients that support IPv6 connections should be able to connect to IPv6 enabled ElastiCache for Redis OSS clusters. In addition, several of the most popular Python and Java clients have been specifically tested and validated to work with all supported network type configurations (IPv4 only, IPv6 only, and Dual Stack)

The following clients have specifically been validated to work with all supported network type configurations for Valkey and Redis OSS.

Validated Clients:

- Redis Py () – 4.1.2
- Lettuce – Version: 6.1.6.RELEASE
- Jedis – Version: 3.6.0

## Best practices for clients (Valkey and Redis OSS)

Learn best practices for common scenarios and follow along with code examples of some of the most popular open source Valkey and Redis OSS client libraries (redis-py, PHPRedis, and Lettuce), as well as best practices for interacting with ElastiCache resources with commonly used open-source Memcached client libraries.

**Topics**


- Large number of connections (Valkey and Redis OSS)
- Cluster client discovery and exponential backoff (Valkey and Redis OSS)
- Configure a client-side timeout (Valkey and Redis OSS)
- Configure a server-side idle timeout (Valkey and Redis OSS)
- Lua scripts

- Storing large composite items (Valkey and Redis OSS)
- Lettuce client configuration (Valkey and Redis OSS)
- Configuring a preferred protocol for dual stack clusters (Valkey and Redis OSS)

## Large number of connections (Valkey and Redis OSS)

Serverless caches and individual ElastiCache for Redis OSS nodes support up to 65,000 concurrent client connections. However, to optimize for performance, we advise that client applications do not constantly operate at that level of connections. Valkey and Redis OSS each have a single-threaded process based on an event loop where incoming client requests are handled sequentially. That means the response time of a given client becomes longer as the number of connected clients increases.

You can take the following set of actions to avoid hitting a connection bottleneck on a Valkey or Redis OSS server:

- Perform read operations from read replicas. This can be done by using the ElastiCache reader endpoints in cluster mode disabled or by using replicas for reads in cluster mode enabled, including a serverless cache.
- Distribute write traffic across multiple primary nodes. You can do this in two ways. You can use a multi-sharded Valkey or Redis OSS cluster with a cluster mode capable client. You could also write to multiple primary nodes in cluster mode disabled with client-side sharding. This is done automatically in a serverless cache.
- Use a connection pool when available in your client library.

In general, creating a TCP connection is a computationally expensive operation compared to typical Valkey or Redis OSS commands. For example, handling a SET/GET request is an order of magnitude faster when reusing an existing connection. Using a client connection pool with a finite size reduces the overhead of connection management. It also bounds the number of concurrent incoming connections from the client application.

The following code example of PHPRedis shows that a new connection is created for each new user request:

```
$redis = new Redis();
if ($redis->connect($HOST, $PORT) != TRUE) {
 //ERROR: connection failed
```

```
 return;
}
$redis->set($key, $value);
unset($redis);
$redis = NULL;
```

We benchmarked this code in a loop on an Amazon Elastic Compute Cloud (Amazon EC2) instance connected to a Graviton2 (m6g.2xlarge) ElastiCache for Redis OSS node. We placed both the client and server at the same Availability Zone. The average latency of the entire operation was 2.82 milliseconds.

When we updated the code and used persistent connections and a connection pool, the average latency of the entire operation was 0.21 milliseconds:

```
$redis = new Redis();
if ($redis->pconnect($HOST, $PORT) != TRUE) {
 // ERROR: connection failed
 return;
}
$redis->set($key, $value);
unset($redis);
$redis = NULL;
```

Required redis.ini configurations:

- `redis.pconnect.pooling_enabled=1`
- `redis.pconnect.connection_limit=10`

The following code is an example of a Redis-py connection pool:

```
conn = Redis(connection_pool=redis.BlockingConnectionPool(host=HOST,
 max_connections=10))
conn.set(key, value)
```

The following code is an example of a Lettuce connection pool:

```
RedisClient client = RedisClient.create(RedisURI.create(HOST, PORT));
GenericObjectPool<StatefulRedisConnection> pool =
 ConnectionPoolSupport.createGenericObjectPool(() -> client.connect(), new
 GenericObjectPoolConfig());
```

```
pool.setMaxTotal(10); // Configure max connections to 10
try (StatefulRedisConnection connection = pool.borrowObject()) {
 RedisCommands syncCommands = connection.sync();
 syncCommands.set(key, value);
}
```

## Cluster client discovery and exponential backoff (Valkey and Redis OSS)

When connecting to an ElastiCache Valkey or Redis OSS cluster in cluster mode enabled, the corresponding client library must be cluster aware. The clients must obtain a map of hash slots to the corresponding nodes in the cluster in order to send requests to the right nodes and avoid the performance overhead of handing cluster redirections. As a result, the client must discover a complete list of slots and the mapped nodes in two different situations:

- The client is initialized and must populate the initial slots configuration
- A MOVED redirection is received from the server, such as in the situation of a failover when all slots served by the former primary node are taken over by the replica, or re-sharding when slots are being moved from the source primary to the target primary node

Client discovery is usually done via issuing a CLUSTER SLOTS or CLUSTER NODES command to the Valkey or Redis OSS server. We recommend the CLUSTER SLOTS method because it returns the set of slot ranges and the associated primary and replica nodes back to the client. This doesn't require additional parsing from the client and is more efficient.

Depending on the cluster topology, the size of the response for the CLUSTER SLOTS command can vary based on the cluster size. Larger clusters with more nodes produce a larger response. As a result, it's important to ensure that the number of clients doing the cluster topology discovery doesn't grow unbounded. For example, when the client application starts up or loses connection from the server and must perform cluster discovery, one common mistake is that the client application fires several reconnection and discovery requests without adding exponential backoff upon retry. This can render the Valkey or Redis OSS server unresponsive for a prolonged period of time, with the CPU utilization at 100%. The outage is prolonged if each CLUSTER SLOTS command must process a large number of nodes in the cluster bus. We have observed multiple client outages in the past due to this behavior across a number of different languages including Python (redis-py-cluster) and Java (Lettuce and Redisson).

In a serverless cache, many of the problems are automatically mitigated because the advertised cluster topology is static and consists of two entries: a write endpoint and a read endpoint. Cluster

discovery is also automatically spread over multiple nodes when using the cache endpoint. The following recommendations are still useful, however.

To mitigate the impact caused by a sudden influx of connection and discovery requests, we recommend the following:

- Implement a client connection pool with a finite size to bound the number of concurrent incoming connections from the client application.
- When the client disconnects from the server due to timeout, retry with exponential backoff with jitter. This helps to avoid multiple clients overwhelming the server at the same time.
- Use the guide at Finding connection endpoints in ElastiCache to find the cluster endpoint to perform cluster discovery. In doing so, you spread the discovery load across all nodes in the cluster (up to 90) instead of hitting a few hardcoded seed nodes in the cluster.

The following are some code examples for exponential backoff retry logic in redis-py, PHPRedis, and Lettuce.

**Backoff logic sample 1: redis-py**

redis-py has a built-in retry mechanism that supports exponential backoff with jitter. If no `retry` parameter is provided, redis-py defaults to `ExponentialWithJitterBackoff` with 3 retries. You can customize the retry behavior through the `retry` and `retry_on_error` arguments when creating a Redis client.

```
from redis.backoff import ExponentialBackoff
from redis.retry import Retry
from redis.client import Redis
from redis.exceptions import ConnectionError, TimeoutError, BusyLoadingError

# Run 3 retries with exponential backoff strategy
retry = Retry(ExponentialBackoff(), 3)

# Redis client with retries
client = Redis(
    host="clustercfg.my-cluster.us-east-1.cache.amazonaws.com",
    port=6379,
    retry=retry,
    retry_on_error=[BusyLoadingError, ConnectionError, TimeoutError],
    ssl=True
)
```

For more information about retry configuration options, see Retry Helpers in the redis-py documentation.

**Backoff logic sample 2: PHPRedis**

PHPRedis has a built-in retry mechanism that retries a (non-configurable) maximum of 10 times. There is a configurable delay between tries (with a jitter from the second retry onwards). For more information, see the following sample code. We've submitted a pull request to natively implement exponential backoff in PHPredis (#1986) that has since been merged and documented. For those on the latest release of PHPRedis, it won't be necessary to implement manually but we've included the reference here for those on previous versions. For now, the following is a code example that configures the delay of the retry mechanism:

```
$timeout = 0.1; // 100 millisecond connection timeout
$retry_interval = 100; // 100 millisecond retry interval
$client = new Redis();
if($client->pconnect($HOST, $PORT, $timeout, NULL, $retry_interval) != TRUE) {
 return; // ERROR: connection failed
}
$client->set($key, $value);
```

**Backoff logic sample 3: Lettuce**

Lettuce has built-in retry mechanisms based on the exponential backoff strategies described in the post Exponential Backoff and Jitter. The following is a code excerpt showing the full jitter approach:

```
public static void main(String[] args)
{
 ClientResources resources = null;
 RedisClient client = null;

 try {
  resources = DefaultClientResources.builder()
    .reconnectDelay(Delay.fullJitter(
   Duration.ofMillis(100),     // minimum 100 millisecond delay
   Duration.ofSeconds(5),      // maximum 5 second delay
   100, TimeUnit.MILLISECONDS) // 100 millisecond base
  ).build();

  client = RedisClient.create(resources, RedisURI.create(HOST, PORT));
```

```
   client.setOptions(ClientOptions.builder()
  .socketOptions(SocketOptions.builder().connectTimeout(Duration.ofMillis(100)).build()) //
  100 millisecond connection timeout
  .timeoutOptions(TimeoutOptions.builder().fixedTimeout(Duration.ofSeconds(5)).build()) //
  5 second command timeout
  .build());

      // use the connection pool from above example
  } finally {
   if (connection != null) {
    connection.close();
   }

   if (client != null){
    client.shutdown();
   }

   if (resources != null){
    resources.shutdown();
   }

  }
}
```

## Configure a client-side timeout (Valkey and Redis OSS)

### Configuring the client-side timeout

Configure the client-side timeout appropriately to allow the server sufficient time to process the request and generate the response. This also allows it to fail fast if the connection to the server can't be established. Certain Valkey or Redis OSS commands can be more computationally expensive than others. For example, Lua scripts or MULTI/EXEC transactions that contain multiple commands that must be run atomically. In general, a higher client-side timeout is recommended to avoid a time out of the client before the response is received from the server, including the following:

- Running commands across multiple keys
- Running MULTI/EXEC transactions or Lua scripts that consist of multiple individual Valkey or Redis OSS commands
- Reading large values
- Performing blocking operations such as BLPOP

In case of a blocking operation such as BLPOP, the best practice is to set the command timeout to a number lower than the socket timeout.

The following are code examples for implementing a client-side timeout in redis-py, PHPRedis, and Lettuce.

**Timeout configuration sample 1: redis-py**

The following is a code example with redis-py:

```
# connect to Redis server with a 100 millisecond timeout
# give every Redis command a 2 second timeout
client = redis.Redis(connection_pool=redis.BlockingConnectionPool(host=HOST,
 max_connections=10,socket_connect_timeout=0.1,socket_timeout=2))

res = client.set("key", "value") # will timeout after 2 seconds
print(res)                       # if there is a connection error

res = client.blpop("list", timeout=1) # will timeout after 1 second
                                      # less than the 2 second socket timeout
print(res)
```

**Timeout config sample 2: PHPRedis**

The following is a code example with PHPRedis:

```
// connect to Redis server with a 100ms timeout
// give every Redis command a 2s timeout
$client = new Redis();
$timeout = 0.1; // 100 millisecond connection timeout
$retry_interval = 100; // 100 millisecond retry interval
$client = new Redis();
if($client->pconnect($HOST, $PORT, 0.1, NULL, 100, $read_timeout=2) != TRUE){
 return; // ERROR: connection failed
}
$client->set($key, $value);

$res = $client->set("key", "value"); // will timeout after 2 seconds
print "$res\n";                      // if there is a connection error

$res = $client->blpop("list", 1); // will timeout after 1 second
print "$res\n";                   // less than the 2 second socket timeout
```

### Timeout config sample 2b: PHPRedis with TLS

When connecting to ElastiCache with in-transit encryption enabled (required for serverless), configure the TLS connection. In the following PHP code example, replace *HOST* with your cluster endpoint and *PORT* with your cluster's port number:

```
// Connect to Amazon ElastiCache with TLS enabled.
$client = new Redis();
$timeout = 2; // 2 second connection timeout.
$read_timeout = 2; // 2 second read timeout.

// TLS connection.
if ($client->connect('tls://'.$HOST, $PORT, $timeout, NULL, 0, $read_timeout) != TRUE)
 {
 return; // ERROR: connection failed.
}

$res = $client->set("key", "value");
print "$res\n";

$res = $client->get("key");
print "$res\n";
```

### Timeout config sample 3: Lettuce

The following is a code example with Lettuce:

```
// connect to Redis server and give every command a 2 second timeout
public static void main(String[] args)
{
 RedisClient client = null;
 StatefulRedisConnection<String, String> connection = null;
 try {
  client = RedisClient.create(RedisURI.create(HOST, PORT));
  client.setOptions(ClientOptions.builder()
 .socketOptions(SocketOptions.builder().connectTimeout(Duration.ofMillis(100)).build()) //
 100 millisecond connection timeout
 .timeoutOptions(TimeoutOptions.builder().fixedTimeout(Duration.ofSeconds(2)).build()) //
 2 second command timeout
 .build());

  // use the connection pool from above example
```

```
  commands.set("key", "value"); // will timeout after 2 seconds
  commands.blpop(1, "list"); // BLPOP with 1 second timeout
 } finally {
  if (connection != null) {
   connection.close();
  }

  if (client != null){
   client.shutdown();
  }
 }
}
```

## Configure a server-side idle timeout (Valkey and Redis OSS)

We have observed cases when a customer's application has a high number of idle clients connected, but isn't actively sending commands. In such scenarios, you can exhaust all 65,000 connections with a high number of idle clients. To avoid such scenarios, configure the timeout setting appropriately on the server via Valkey and Redis OSS parameters. This ensures that the server actively disconnects idle clients to avoid an increase in the number of connections. This setting is not available on serverless caches.

## Lua scripts

Valkey and Redis OSS supports more than 200 commands, including those to run Lua scripts. However, when it comes to Lua scripts, there are several pitfalls that can affect memory and availability of Valkey or Redis OSS.

### Unparameterized Lua scripts

Each Lua script is cached on the Valkey or Redis OSS server before it runs. Unparameterized Lua scripts are unique, which can lead to the Valkey or Redis OSS server storing a large number of Lua scripts and consuming more memory. To mitigate this, ensure that all Lua scripts are parameterized and regularly perform SCRIPT FLUSH to clean up cached Lua scripts if needed.

Also be aware that keys must be provided. If a value for the KEY parameter is not provided, the script will fail. For example, this will not work:

```
serverless-test-lst4hg.serverless.use1.cache.amazonaws.com:6379> eval 'return "Hello
 World"' 0
```

```
(error) ERR Lua scripts without any input keys are not supported.
```

This will work:

```
serverless-test-lst4hg.serverless.use1.cache.amazonaws.com:6379> eval 'return
 redis.call("get", KEYS[1])' 1 mykey-2
"myvalue-2"
```

The following example shows how to use parameterized scripts. First, we have an example of an unparameterized approach that results in three different cached Lua scripts and is not recommended:

```
eval "return redis.call('set','key1','1')" 0
eval "return redis.call('set','key2','2')" 0
eval "return redis.call('set','key3','3')" 0
```

Instead, use the following pattern to create a single script that can accept passed parameters:

```
eval "return redis.call('set',KEYS[1],ARGV[1])" 1 key1 1
eval "return redis.call('set',KEYS[1],ARGV[1])" 1 key2 2
eval "return redis.call('set',KEYS[1],ARGV[1])" 1 key3 3
```

**Long-running Lua scripts**

Lua scripts can run multiple commands atomically, so it can take longer to complete than a regular Valkey or Redis OSS command. If the Lua script only runs read-only operations, you can stop it in the middle. However, as soon as the Lua script performs a write operation, it becomes unkillable and must run to completion. A long-running Lua script that is mutating can cause the Valkey or Redis OSS server to be unresponsive for a long time. To mitigate this issue, avoid long-running Lua scripts and test the script out in a pre-production environment.

**Lua script with stealth writes**

There are a few ways a Lua script can continue to write new data into Valkey or Redis OSS even when Valkey or Redis OSS is over `maxmemory`:

- The script starts when the Valkey or Redis OSS server is below `maxmemory`, and contains multiple write operations inside
- The script's first write command isn't consuming memory (such as DEL), followed by more write operations that consume memory

- You can mitigate this problem by configuring a proper eviction policy in Valkey or Redis OSS server other than `noeviction`. This allows Redis OSS to evict items and free up memory in between Lua scripts.

## Storing large composite items (Valkey and Redis OSS)

In some scenarios, an application may store large composite items in Valkey or Redis OSS (such as a multi-GB hash dataset). This is not a recommended practice because it often leads to performance problems in Valkey or Redis OSS. For example, the client can do a HGETALL command to retrieve the entire multi GB hash collection. This can generate significant memory pressure to the Valkey or Redis OSS server buffering the large item in the client output buffer. Also, for slot migration in cluster mode, ElastiCache doesn't migrate slots that contain items with serialized size that is larger than 256 MB.

To solve the large item problems, we have the following recommendations:

- Break up the large composite item into multiple smaller items. For example, break up a large hash collection into individual key-value fields with a key name scheme that appropriately reflects the collection, such as using a common prefix in the key name to identify the collection of items. If you must access multiple fields in the same collection atomically, you can use the MGET command to retrieve multiple key-values in the same command.
- If you evaluated all options and still can't break up the large collection dataset, try to use commands that operate on a subset of the data in the collection instead of the entire collection. Avoid having a use case that requires you to atomically retrieve the entire multi-GB collection in the same command. One example is using HGET or HMGET commands instead of HGETALL on hash collections.

## Lettuce client configuration (Valkey and Redis OSS)

This section describes the recommended Java and Lettuce configuration options, and how they apply to ElastiCache clusters.

The recommendations in this section were tested with Lettuce version 6.2.2.

**Topics**


- Example: Lettuce config for cluster mode, TLS enabled
- Example: Lettuce config for cluster mode disabled, TLS enabled

- Distributing reads to replicas (cluster mode disabled)

**Java DNS cache TTL**

The Java virtual machine (JVM) caches DNS name lookups. When the JVM resolves a hostname to an IP address, it caches the IP address for a specified period of time, known as the *time-to-live* (TTL).

The choice of TTL value is a trade-off between latency and responsiveness to change. With shorter TTLs, DNS resolvers notice updates in the cluster's DNS faster. This can make your application respond faster to replacements or other workflows that your cluster undergoes. However, if the TTL is too low, it increases the query volume, which can increase the latency of your application. While there is no correct TTL value, it's worth considering the length of time that you can afford to wait for a change to take effect when setting your TTL value.

Because ElastiCache nodes use DNS name entries that might change, we recommend that you configure your JVM with a low TTL of 5 to 10 seconds. This ensures that when a node's IP address changes, your application will be able to receive and use the resource's new IP address by requerying the DNS entry.

On some Java configurations, the JVM default TTL is set so that it will never refresh DNS entries until the JVM is restarted.

For details on how to set your JVM TTL, see How to set the JVM TTL.

**Lettuce version**

We recommend using Lettuce version 6.2.2 or later.

**Endpoints**

When you're using cluster mode enabled clusters, set the `redisUri` to the cluster configuration endpoint. The DNS lookup for this URI returns a list of all available nodes in the cluster, and is randomly resolved to one of them during the cluster initialization. For more details about how topology refresh works, see *dynamicRefreshSources* later in this topic.

**SocketOption**

For more information about the configuration options in this section, see the Lettuce core API reference on the Lettuce Javadoc website.

Enable KeepAlive. Enabling this option reduces the need to handle failed connections during command runtime.

Ensure that you set the Connection timeout based on your application requirements and workload. For more information, see the Timeouts section later in this topic.

**ClusterClientOption: Cluster Mode Enabled client options**

Enable AutoReconnect when connection is lost.

Set CommandTimeout. For more details, see the Timeouts section later in this topic.

Set nodeFilter to filter out failed nodes from the topology. Lettuce saves all nodes that are found in the 'cluster nodes' output (including nodes with PFAIL/FAIL status) in the client's 'partitions' (also known as shards). During the process of creating the cluster topology, it attempts to connect to all the partition nodes. This Lettuce behavior of adding failed nodes can cause connection errors (or warnings) when nodes are getting replaced for any reason.

For example, after a failover is finished and the cluster starts the recovery process, while the clusterTopology is getting refreshed, the cluster bus nodes map has a short period of time that the down node is listed as a FAIL node, before it's completely removed from the topology. During this period, the Lettuce client considers it a healthy node and continually connects to it. This causes a failure after retrying is exhausted.

For example:

```
final ClusterClientOptions clusterClientOptions =
    ClusterClientOptions.builder()
    ... // other options
    .nodeFilter(it ->
        ! (it.is(RedisClusterNode.NodeFlag.FAIL)
        || it.is(RedisClusterNode.NodeFlag.EVENTUAL_FAIL)
        || it.is(RedisClusterNode.NodeFlag.HANDSHAKE)
        || it.is(RedisClusterNode.NodeFlag.NOADDR)))
    .validateClusterNodeMembership(false)
    .build();
redisClusterClient.setOptions(clusterClientOptions);
```

**Note**

Node filtering is best used with DynamicRefreshSources set to true. Otherwise, if the topology view is taken from a single problematic seed node, that sees a primary node

of some shard as failing, it will filter out this primary node, which will result in slots not being covered. Having multiple seed nodes (when DynamicRefreshSources is true) reduces the likelihood of this issue, since at least some of the seed nodes should have an updated topology view after a failover with the newly promoted primary.

**ClusterTopologyRefreshOptions: Options to control the cluster topology refreshing of the Cluster Mode Enabled client**

**Note**

Cluster mode disabled clusters don't support the cluster discovery commands and aren't compatible with all clients dynamic topology discovery functionality.
Cluster mode disabled with ElastiCache isn't compatible with Lettuce's `MasterSlaveTopologyRefresh`. Instead, for cluster mode disabled you can configure a `StaticMasterReplicaTopologyProvider` and provide the cluster read and write endpoints.
For more information on connecting to cluster mode disabled clusters, see Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console).
If you wish to use Lettuce's dynamic topology discovery functionality, then you can create a cluster mode enabled cluster with the same shard configuration as your existing cluster. However, for cluster mode enabled clusters we recommend configuring at least 3 shards with at least one 1 replica to support fast failover.

Enable enablePeriodicRefresh. This enables periodic cluster topology updates so that the client updates the cluster topology in the intervals of the refreshPeriod (default: 60 seconds). When it's disabled, the client updates the cluster topology only when errors occur when it attempts to run commands against the cluster.

With this option enabled, you can reduce the latency that's associated with refreshing the cluster topology by adding this job to a background task. While topology refreshment is performed in a background job, it can be somewhat slow for clusters with many nodes. This is because all nodes are being queried for their views to get the most updated cluster view. If you run a large cluster, you might want to increase the period.

Enable enableAllAdaptiveRefreshTriggers. This enables adaptive topology refreshing that uses all triggers: MOVED_REDIRECT, ASK_REDIRECT, PERSISTENT_RECONNECTS, UNCOVERED_SLOT,

UNKNOWN_NODE. Adaptive refresh triggers initiate topology view updates based on events that happen during Valkey or Redis OSS cluster operations. Enabling this option leads to an immediate topology refresh when one of the preceding triggers occur. Adaptive triggered refreshes are rate-limited using a timeout because events can happen on a large scale (default timeout between updates: 30).

Enable closeStaleConnections. This enables closing stale connections when refreshing the cluster topology. It only comes into effect if ClusterTopologyRefreshOptions.isPeriodicRefreshEnabled() is true. When it's enabled, the client can close stale connections and create new ones in the background. This reduces the need to handle failed connections during command runtime.

Enable dynamicRefreshSources. We recommend enabling dynamicRefreshSources for small clusters, and disabling it for large clusters. dynamicRefreshSources enables discovering cluster nodes from the provided seed node (for example, cluster configuration endpoint). It uses all the discovered nodes as sources for refreshing the cluster topology.

Using dynamic refresh queries all discovered nodes for the cluster topology and attempts to choose the most accurate cluster view. If it's set to false, only the initial seed nodes are used as sources for topology discovery, and the number of clients are obtained only for the initial seed nodes. When it's disabled, if the cluster configuration endpoint is resolved to a failed node, trying to refresh the cluster view fails and leads to exceptions. This scenario can happen because it takes some time until a failed node's entry is removed from the cluster configuration endpoint. Therefore, the configuration endpoint can still be randomly resolved to a failed node for a short period of time.

When it's enabled, however, we use all of the cluster nodes that are received from the cluster view to query for their current view. Because we filter out failed nodes from that view, the topology refresh will be successful. However, when dynamicRefreshSources is true, Lettuce queries all nodes to get the cluster view, and then compares the results. So it can be expensive for clusters with a lot of nodes. We suggest that you turn off this feature for clusters with many nodes.

```
final ClusterTopologyRefreshOptions topologyOptions =
    ClusterTopologyRefreshOptions.builder()
    .enableAllAdaptiveRefreshTriggers()
    .enablePeriodicRefresh()
    .dynamicRefreshSources(true)
    .build();
```

**ClientResources**

Starting with Lettuce 6.7, Lettuce uses Netty's `DnsAddressResolverGroup` by default for non-blocking DNS resolution. You no longer need to configure a custom DNS resolver. If you are using Lettuce 6.6 or earlier, configure `DnsResolver` with `DirContextDnsResolver` to enable proper DNS-based node discovery.

Configure reconnectDelay with exponential backoff and full jitter. Lettuce has built-in retry mechanisms based on the exponential backoff strategies.. For details, see Exponential Backoff and Jitter on the AWS Architecture Blog. For more information about the importance of having a retry backoff strategy, see the backoff logic sections of the Best practices blog post on the AWS Database Blog.

```
ClientResources clientResources = DefaultClientResources.builder()
    .reconnectDelay(
        Delay.fullJitter(
            Duration.ofMillis(100),     // minimum 100 millisecond delay
            Duration.ofSeconds(10),      // maximum 10 second delay
            100, TimeUnit.MILLISECONDS)) // 100 millisecond base
    .build();
```

**Timeouts**

Use a lower connect timeout value than your command timeout. Lettuce uses lazy connection establishment. So if the connect timeout is higher than the command timeout, you can have a period of persistent failure after a topology refresh if Lettuce tries to connect to an unhealthy node and the command timeout is always exceeded.

Use a dynamic command timeout for different commands. We recommend that you set the command timeout based on the command expected duration. For example, use a longer timeout for commands that iterate over several keys, such as FLUSHDB, FLUSHALL, KEYS, SMEMBERS, or Lua scripts. Use shorter timeouts for single key commands, such as SET, GET, and HSET.

> **Note**
>
> Timeouts that are configured in the following example are for tests that ran SET/GET commands with keys and values up to 20 bytes long. The processing time can be longer when the commands are complex or the keys and values are larger. You should set the timeouts based on the use case of your application.

```
private static final Duration META_COMMAND_TIMEOUT = Duration.ofMillis(1000);
private static final Duration DEFAULT_COMMAND_TIMEOUT = Duration.ofMillis(250);
// Socket connect timeout should be lower than command timeout for Lettuce
private static final Duration CONNECT_TIMEOUT = Duration.ofMillis(100);

SocketOptions socketOptions = SocketOptions.builder()
    .connectTimeout(CONNECT_TIMEOUT)
    .build();


class DynamicClusterTimeout extends TimeoutSource {
     private static final Set<ProtocolKeyword> META_COMMAND_TYPES =
 ImmutableSet.<ProtocolKeyword>builder()
          .add(CommandType.FLUSHDB)
          .add(CommandType.FLUSHALL)
          .add(CommandType.CLUSTER)
          .add(CommandType.INFO)
          .add(CommandType.KEYS)
          .build();

    private final Duration defaultCommandTimeout;
    private final Duration metaCommandTimeout;

    DynamicClusterTimeout(Duration defaultTimeout, Duration metaTimeout)
    {
        defaultCommandTimeout = defaultTimeout;
        metaCommandTimeout = metaTimeout;
    }

    @Override
    public long getTimeout(RedisCommand<?, ?, ?> command) {
        if (META_COMMAND_TYPES.contains(command.getType())) {
            return metaCommandTimeout.toMillis();
        }
        return defaultCommandTimeout.toMillis();
    }
}

// Use a dynamic timeout for commands, to avoid timeouts during
// cluster management and slow operations.
TimeoutOptions timeoutOptions = TimeoutOptions.builder()
.timeoutSource(
    new DynamicClusterTimeout(DEFAULT_COMMAND_TIMEOUT, META_COMMAND_TIMEOUT))
```

```
.build();
```

### Example: Lettuce config for cluster mode, TLS enabled

> **Note**
>
> Timeouts in the following example are for tests that ran SET/GET commands with keys and values up to 20 bytes long. The processing time can be longer when the commands are complex or the keys and values are larger. You should set the timeouts based on the use case of your application.

```
// Set DNS cache TTL
public void setJVMProperties() {
    java.security.Security.setProperty("networkaddress.cache.ttl", "10");
}

private static final Duration META_COMMAND_TIMEOUT = Duration.ofMillis(1000);
private static final Duration DEFAULT_COMMAND_TIMEOUT = Duration.ofMillis(250);
// Socket connect timeout should be lower than command timeout for Lettuce
private static final Duration CONNECT_TIMEOUT = Duration.ofMillis(100);

// Create RedisURI from the cluster configuration endpoint
clusterConfigurationEndpoint = <cluster-configuration-endpoint> // TODO: add your
 cluster configuration endpoint
final RedisURI redisUriCluster =
    RedisURI.Builder.redis(clusterConfigurationEndpoint)
        .withPort(6379)
        .withSsl(true)
        .build();

// Configure the client's resources
ClientResources clientResources = DefaultClientResources.builder()
    .reconnectDelay(
        Delay.fullJitter(
            Duration.ofMillis(100),     // minimum 100 millisecond delay
            Duration.ofSeconds(10),      // maximum 10 second delay
            100, TimeUnit.MILLISECONDS)) // 100 millisecond base
    .addressResolverGroup(new DirContextDnsResolver())
    .build();

// Create a cluster client instance with the URI and resources
```

```
RedisClusterClient redisClusterClient =
    RedisClusterClient.create(clientResources, redisUriCluster);

// Use a dynamic timeout for commands, to avoid timeouts during
// cluster management and slow operations.
class DynamicClusterTimeout extends TimeoutSource {
     private static final Set<ProtocolKeyword> META_COMMAND_TYPES =
 ImmutableSet.<ProtocolKeyword>builder()
          .add(CommandType.FLUSHDB)
          .add(CommandType.FLUSHALL)
          .add(CommandType.CLUSTER)
          .add(CommandType.INFO)
          .add(CommandType.KEYS)
          .build();

    private final Duration metaCommandTimeout;
    private final Duration defaultCommandTimeout;

    DynamicClusterTimeout(Duration defaultTimeout, Duration metaTimeout)
    {
        defaultCommandTimeout = defaultTimeout;
        metaCommandTimeout = metaTimeout;
    }

    @Override
    public long getTimeout(RedisCommand<?, ?, ?> command) {
        if (META_COMMAND_TYPES.contains(command.getType())) {
            return metaCommandTimeout.toMillis();
        }
        return defaultCommandTimeout.toMillis();
    }
}

TimeoutOptions timeoutOptions = TimeoutOptions.builder()
    .timeoutSource(new DynamicClusterTimeout(DEFAULT_COMMAND_TIMEOUT,
 META_COMMAND_TIMEOUT))
     .build();

// Configure the topology refreshment options
final ClusterTopologyRefreshOptions topologyOptions =
    ClusterTopologyRefreshOptions.builder()
    .enableAllAdaptiveRefreshTriggers()
    .enablePeriodicRefresh()
    .dynamicRefreshSources(true)
```

```
    .build();

// Configure the socket options
final SocketOptions socketOptions =
    SocketOptions.builder()
    .connectTimeout(CONNECT_TIMEOUT)
    .keepAlive(true)
    .build();

// Configure the client's options
final ClusterClientOptions clusterClientOptions =
    ClusterClientOptions.builder()
    .topologyRefreshOptions(topologyOptions)
    .socketOptions(socketOptions)
    .autoReconnect(true)
    .timeoutOptions(timeoutOptions)
    .nodeFilter(it ->
        ! (it.is(RedisClusterNode.NodeFlag.FAIL)
        || it.is(RedisClusterNode.NodeFlag.EVENTUAL_FAIL)
        || it.is(RedisClusterNode.NodeFlag.NOADDR)))
    .validateClusterNodeMembership(false)
    .build();

redisClusterClient.setOptions(clusterClientOptions);

// Get a connection
final StatefulRedisClusterConnection<String, String> connection =
    redisClusterClient.connect();

// Get cluster sync/async commands
RedisAdvancedClusterCommands<String, String> sync = connection.sync();
RedisAdvancedClusterAsyncCommands<String, String> async = connection.async();
```

### Example: Lettuce config for cluster mode disabled, TLS enabled

> **Note**
>
> Timeouts in the following example are for tests that ran SET/GET commands with keys and values up to 20 bytes long. The processing time can be longer when the commands are complex or the keys and values are larger. You should set the timeouts based on the use case of your application.

```
// Set DNS cache TTL
public void setJVMProperties() {
    java.security.Security.setProperty("networkaddress.cache.ttl", "10");
}

private static final Duration META_COMMAND_TIMEOUT = Duration.ofMillis(1000);
private static final Duration DEFAULT_COMMAND_TIMEOUT = Duration.ofMillis(250);
// Socket connect timeout should be lower than command timeout for Lettuce
private static final Duration CONNECT_TIMEOUT = Duration.ofMillis(100);

// Create RedisURI from the primary/reader endpoint
clusterEndpoint = <primary/reader-endpoint> // TODO: add your node endpoint
RedisURI redisUriStandalone =

 RedisURI.Builder.redis(clusterEndpoint).withPort(6379).withSsl(true).withDatabase(0).build();

ClientResources clientResources =
    DefaultClientResources.builder()
        .addressResolverGroup(new DirContextDnsResolver())
        .reconnectDelay(
            Delay.fullJitter(
                Duration.ofMillis(100), // minimum 100 millisecond delay
                Duration.ofSeconds(10), // maximum 10 second delay
                100,
                TimeUnit.MILLISECONDS)) // 100 millisecond base
        .build();

// Use a dynamic timeout for commands, to avoid timeouts during
// slow operations.
class DynamicTimeout extends TimeoutSource {
     private static final Set<ProtocolKeyword> META_COMMAND_TYPES =
 ImmutableSet.<ProtocolKeyword>builder()
          .add(CommandType.FLUSHDB)
          .add(CommandType.FLUSHALL)
          .add(CommandType.INFO)
          .add(CommandType.KEYS)
          .build();

    private final Duration metaCommandTimeout;
    private final Duration defaultCommandTimeout;

    DynamicTimeout(Duration defaultTimeout, Duration metaTimeout)
    {
```

```
        defaultCommandTimeout = defaultTimeout;
        metaCommandTimeout = metaTimeout;
    }

    @Override
    public long getTimeout(RedisCommand<?, ?, ?> command) {
        if (META_COMMAND_TYPES.contains(command.getType())) {
            return metaCommandTimeout.toMillis();
        }
        return defaultCommandTimeout.toMillis();
    }
}

TimeoutOptions timeoutOptions = TimeoutOptions.builder()
    .timeoutSource(new DynamicTimeout(DEFAULT_COMMAND_TIMEOUT, META_COMMAND_TIMEOUT))
     .build();

final SocketOptions socketOptions =
    SocketOptions.builder().connectTimeout(CONNECT_TIMEOUT).keepAlive(true).build();

ClientOptions clientOptions =

 ClientOptions.builder().timeoutOptions(timeoutOptions).socketOptions(socketOptions).build();

RedisClient redisClient = RedisClient.create(clientResources, redisUriStandalone);
redisClient.setOptions(clientOptions);
```

**Distributing reads to replicas (cluster mode disabled)**

For cluster mode disabled clusters with replicas, you can use Lettuce's `StaticMasterReplicaTopologyProvider` with the `ReadFrom` setting to direct writes to the primary node and distribute reads across replicas. This improves read throughput by offloading reads from the primary.

Use the primary endpoint for writes and the reader endpoint for reads. The reader endpoint automatically load-balances connections across available replicas.

```
// Primary endpoint for writes, reader endpoint for read distribution
RedisURI primaryURI = RedisURI.Builder.redis("primary-endpoint.cache.amazonaws.com")
    .withPort(6379).withSsl(true).build();
RedisURI readerURI = RedisURI.Builder.redis("reader-endpoint.cache.amazonaws.com")
    .withPort(6379).withSsl(true).build();
```

```
RedisClient redisClient = RedisClient.create(clientResources);
redisClient.setOptions(clientOptions);

// Open a primary-replica connection using the StaticMasterReplicaTopologyProvider
StatefulRedisMasterReplicaConnection<String, String> connection =
    MasterReplica.connect(redisClient, StringCodec.UTF8, Arrays.asList(primaryURI,
 readerURI));

// Direct reads to replicas, writes always go to the primary
connection.setReadFrom(ReadFrom.REPLICA_PREFERRED);
```

The following `ReadFrom` options are available:

- `REPLICA_PREFERRED` – Reads from replicas when available and falls back to the primary (recommended for most use cases).
- `REPLICA` – Reads only from replicas. Fails if no replica is available.
- `PRIMARY_PREFERRED` – Reads from the primary, falls back to replicas.
- `ANY_REPLICA` – Reads from any replica node.
- `PRIMARY` – Reads only from the primary node. Use when strong consistency is required.

> **Note**
>
> Data read from replicas might be slightly stale because replication is asynchronous. If your application requires strong consistency, use `ReadFrom.PRIMARY` or `ReadFrom.PRIMARY_PREFERRED`.

## Configuring a preferred protocol for dual stack clusters (Valkey and Redis OSS)

For cluster mode enabled Valkey or Redis OSS clusters, you can control the protocol clients will use to connect to the nodes in the cluster with the IP Discovery parameter. The IP Discovery parameter can be set to either IPv4 or IPv6.

For Valkey or Redis OSS clusters, the IP discovery parameter sets the IP protocol used in the cluster slots (), cluster shards (), and cluster nodes () output. These commands are used by clients to discover the cluster topology. Clients use the IPs in theses commands to connect to the other nodes in the cluster.

Changing the IP Discovery will not result in any downtime for connected clients. However, the changes will take some time to propagate. To determine when the changes have completely propagated for a Valkey or Redis OSS Cluster, monitor the output of `cluster slots`. Once all of the nodes returned by the cluster slots command report IPs with the new protocol the changes have finished propagating.

Example with Redis-Py:

```
cluster = RedisCluster(host="xxxx", port=6379)
target_type = IPv6Address # Or IPv4Address if changing to IPv4

nodes = set()
while len(nodes) == 0 or not all((type(ip_address(host)) is target_type) for host in
 nodes):
    nodes = set()

   # This refreshes the cluster topology and will discovery any node updates.
   # Under the hood it calls cluster slots
    cluster.nodes_manager.initialize()
    for node in cluster.get_nodes():
        nodes.add(node.host)
    self.logger.info(nodes)

    time.sleep(1)
```

Example with Lettuce:

```
RedisClusterClient clusterClient = RedisClusterClient.create(RedisURI.create("xxxx",
 6379));

Class targetProtocolType = Inet6Address.class; // Or Inet4Address.class if you're
 switching to IPv4

Set<String> nodes;

do {
   // Check for any changes in the cluster topology.
   // Under the hood this calls cluster slots
    clusterClient.refreshPartitions();
    Set<String> nodes = new HashSet<>();

    for (RedisClusterNode node : clusterClient.getPartitions().getPartitions()) {
```

```
        nodes.add(node.getUri().getHost());
    }

    Thread.sleep(1000);
} while (!nodes.stream().allMatch(node -> {
            try {
                return finalTargetProtocolType.isInstance(InetAddress.getByName(node));
            } catch (UnknownHostException ignored) {}
            return false;
}));
```

## Best practices for clients (Memcached)

Learn best practices for common scenarios with ElastiCache for Memcached clusters.

**Topics**

- Configuring your ElastiCache client for efficient load balancing (Memcached)
- Validated clients with Memcached
- Configuring a preferred protocol for dual stack clusters (Memcached)

### Configuring your ElastiCache client for efficient load balancing (Memcached)

> **Note**
>
> This section applies to node-based multi-node Memcached clusters.

To effectively use multiple ElastiCache Memcached nodes, you need to be able to spread your cache keys across the nodes. A simple way to load balance a cluster with *n* nodes is to calculate the hash of the object's key and mod the result by *n*: `hash(key) mod n`. The resulting value (0 through *n*–1) is the number of the node where you place the object.

This approach is simple and works well as long as the number of nodes (*n*) is constant. However, whenever you add or remove a node from the cluster, the number of keys that need to be moved is *(n - 1) / n* (where *n* is the new number of nodes). Thus, this approach results in a large number of keys being moved, which translates to a large number of initial cache misses, especially as the number of nodes gets large. Scaling from 1 to 2 nodes results in (2-1) / 2 (50 percent) of the keys being moved, the best case. Scaling from 9 to 10 nodes results in (10–1)/10 (90 percent) of the

keys being moved. If you're scaling up due to a spike in traffic, you don't want to have a large number of cache misses. A large number of cache misses results in hits to the database, which is already overloaded due to the spike in traffic.

The solution to this dilemma is consistent hashing. Consistent hashing uses an algorithm such that whenever a node is added or removed from a cluster, the number of keys that must be moved is roughly *1 / n* (where *n* is the new number of nodes). Scaling from 1 to 2 nodes results in 1/2 (50 percent) of the keys being moved, the worst case. Scaling from 9 to 10 nodes results in 1/10 (10 percent) of the keys being moved.

As the user, you control which hashing algorithm is used for multi-node clusters. We recommend that you configure your clients to use consistent hashing. Fortunately, there are many Memcached client libraries in most popular languages that implement consistent hashing. Check the documentation for the library you are using to see if it supports consistent hashing and how to implement it.

If you are working in Java, PHP, or .NET, we recommend you use one of the Amazon ElastiCache client libraries.

**Consistent Hashing Using Java**

The ElastiCache Memcached Java client is based on the open-source spymemcached Java client, which has consistent hashing capabilities built in. The library includes a KetamaConnectionFactory class that implements consistent hashing. By default, consistent hashing is turned off in spymemcached.

For more information, see the KetamaConnectionFactory documentation at KetamaConnectionFactory.

**Consistent hashing using PHP with Memcached**

The ElastiCache Memcached PHP client is a wrapper around the built-in Memcached PHP library. By default, consistent hashing is turned off by the Memcached PHP library.

Use the following code to turn on consistent hashing.

```
$m = new Memcached();
$m->setOption(Memcached::OPT_DISTRIBUTION, Memcached::DISTRIBUTION_CONSISTENT);
```

In addition to the preceding code, we recommend that you also turn `memcached.sess_consistent_hash` on in your php.ini file.

For more information, see the run-time configuration documentation for Memcached PHP at http://php.net/manual/en/memcached.configuration.php. Note specifically the `memcached.sess_consistent_hash` parameter.

**Consistent hashing using .NET with Memcached**

The ElastiCache Memcached .NET client is a wrapper around Enyim Memcached. By default, consistent hashing is turned on by the Enyim Memcached client.

For more information, see the `memcached/locator` documentation at https://github.com/enyim/EnyimMemcached/wiki/MemcachedClient-Configuration#user-content-memcachedlocator.

## Validated clients with Memcached

The following clients have specifically been validated to work with all supported network type configurations for Memcached.

Validated Clients:

- AWS ElastiCache Cluster Client Memcached for Php – Version *3.6.2
- AWS ElastiCache Cluster Client Memcached for Java – Latest master on Github

## Configuring a preferred protocol for dual stack clusters (Memcached)

For Memcached clusters you can control the protocol clients will use to connect to the nodes in the cluster with the IP Discovery parameter. The IP Discovery parameter can be set to either IPv4 or IPv6.

The IP discovery parameter controls the IP protocol used in the config get cluster output. Which in turn will determine the IP protocol used by clients which support auto-discovery for ElastiCache for Memcached clusters.

Changing the IP Discovery will not result in any downtime for connected clients. However, the changes will take some time to propagate.

Monitor the output of `getAvailableNodeEndPoints` for Java and for Php monitor the output of `getServerList`. Once the output of these functions reports IPs for all of the nodes in the cluster that use the updated protocol, the changes have finished propagating.

Java Example:

```
MemcachedClient client = new MemcachedClient(new InetSocketAddress("xxxx", 11211));

Class targetProtocolType = Inet6Address.class; // Or Inet4Address.class if you're
 switching to IPv4

Set<String> nodes;

do {
    nodes =
 client.getAvailableNodeEndPoints().stream().map(NodeEndPoint::getIpAddress).collect(Collectors

    Thread.sleep(1000);
} while (!nodes.stream().allMatch(node -> {
            try {
                return finalTargetProtocolType.isInstance(InetAddress.getByName(node));
            } catch (UnknownHostException ignored) {}
            return false;
        }));
```

Php Example:

```
$client = new Memcached;
$client->setOption(Memcached::OPT_CLIENT_MODE, Memcached::DYNAMIC_CLIENT_MODE);
$client->addServer("xxxx", 11211);

$nodes = [];
$target_ips_count = 0;
do {
    # The PHP memcached client only updates the server list if the polling interval has
 expired and a
    # command is sent
    $client->get('test');

    $nodes = $client->getServerList();

    sleep(1);
    $target_ips_count = 0;

    // For IPv4 use FILTER_FLAG_IPV4
    $target_ips_count = count(array_filter($nodes, function($node) { return
 filter_var($node["ipaddress"], FILTER_VALIDATE_IP, FILTER_FLAG_IPV6); }));
```

```
} while (count($nodes) !== $target_ips_count);
```

Any existing client connections that were created before the IP Discovery was updated will still be connected using the old protocol. All of the validated clients will automatically reconnect to the cluster using the new IP protocol once the changes are detected in the output of the cluster discovery commands. However, this depends on the implementation of the client.

## TLS enabled dual stack ElastiCache clusters

When TLS is enabled for ElastiCache clusters the cluster discovery functions (`cluster slots`, `cluster shards`, and `cluster nodes` for Redis) or `config get cluster` for Memcached return hostnames instead of IPs. The hostnames are then used instead of IPs to connect to the ElastiCache cluster and perform a TLS handshake. This means that clients won't be affected by the IP Discovery parameter. For TLS enabled clusters the IP Discovery parameter has no effect on the preferred IP protocol. Instead, the IP protocol used will be determined by which IP protocol the client prefers when resolving DNS hostnames.

**Java clients**

When connecting from a Java environment that supports both IPv4 and IPv6, Java will by default prefer IPv4 over IPv6 for backwards compatibility. However, the IP protocol preference is configurable through the JVM arguments. To prefer IPv4, the JVM accepts `-Djava.net.preferIPv4Stack=true` and to prefer IPv6 set `-Djava.net.preferIPv6Stack=true`. Setting `-Djava.net.preferIPv4Stack=true` means that the JVM will no longer make any IPv6 connections. **For Valkey or Redis OSS, this includes those to other non-Valkey and non-Redis OSS applications.**

**Host Level Preferences**

In general, if the client or client runtime don't provide configuration options for setting an IP protocol preference, when performing DNS resolution, the IP protocol will depend on the host's configuration. By default, most hosts prefer IPv6 over IPv4 but this preference can be configured at the host level. This will affect all DNS requests from that host, not just those to ElastiCache clusters.

**Linux hosts**

For Linux, an IP protocol preference can be configured by modifying the `gai.conf` file. The `gai.conf` file can be found under `/etc/gai.conf`. If there is no `gai.conf` specified then an example one should be available under `/usr/share/doc/glibc-common-x.xx/gai.conf`

which can be copied to `/etc/gai.conf` and then the default configuration should be un-commented. To update the configuration to prefer IPv4 when connecting to an ElastiCache cluster update the precedence for the CIDR range encompassing the cluster IPs to be above the precedence for default IPv6 connections. By default IPv6 connections have a precedence of 40. For example, assuming the cluster is located in a subnet with the CIDR 172.31.0.0:0/16, the configuration below would cause clients to prefer IPv4 connections to that cluster.

```
label ::1/128       0
label ::/0          1
label 2002::/16     2
label ::/96         3
label ::ffff:0:0/96 4
label fec0::/10     5
label fc00::/7      6
label 2001:0::/32   7
label ::ffff:172.31.0.0/112 8
#
#    This default differs from the tables given in RFC 3484 by handling
#    (now obsolete) site-local IPv6 addresses and Unique Local Addresses.
#    The reason for this difference is that these addresses are never
#    NATed while IPv4 site-local addresses most probably are.  Given
#    the precedence of IPv6 over IPv4 (see below) on machines having only
#    site-local IPv4 and IPv6 addresses a lookup for a global address would
#    see the IPv6 be preferred.  The result is a long delay because the
#    site-local IPv6 addresses cannot be used while the IPv4 address is
#    (at least for the foreseeable future) NATed.  We also treat Teredo
#    tunnels special.
#
# precedence  <mask>   <value>
#    Add another rule to the RFC 3484 precedence table.  See section 2.1
#    and 10.3 in RFC 3484.  The default is:
#
precedence  ::1/128       50
precedence  ::/0          40
precedence  2002::/16     30
precedence ::/96          20
precedence ::ffff:0:0/96  10
precedence ::ffff:172.31.0.0/112 100
```

More details on `gai.conf` are available on the Linux man page

### Windows hosts

The process for Windows hosts is similar. For Windows hosts you can run `netsh interface ipv6 set prefix CIDR_CONTAINING_CLUSTER_IPS PRECEDENCE LABEL`. This has the same effect as modifying the `gai.conf` file on Linux hosts.

This will update the preference policies to prefer IPv4 connections over IPv6 connections for the specified CIDR range. For example, assuming that the cluster is in a subnet with the 172.31.0.0:0/16 CIDR executing `netsh interface ipv6 set prefix ::ffff:172.31.0.0:0/112 100 15` would result in the following precedence table which would cause clients to prefer IPv4 when connecting to the cluster.

```
C:\Users\Administrator>netsh interface ipv6 show prefixpolicies
Querying active state...

Precedence Label Prefix
---------- ----- --------------------------------
100 15 ::ffff:172.31.0.0:0/112
20 4 ::ffff:0:0/96
50 0 ::1/128
40 1 ::/0
30 2 2002::/16
5 5 2001::/32
3 13 fc00::/7
1 11 fec0::/10
1 12 3ffe::/16
1 3 ::/96
```

# Managing reserved memory for Valkey and Redis OSS

Each host type has an amount of memory dedicated for Valkey and Redis OSS. This is the advertised memory. The `reserved-memory-percent` parameter sets aside part of this memory for nondata uses:

- Client output buffers on replicas
- Memory lost to fragmentation
- Copy-on-write memory used by forked bgsave

The `maxmemory` value equals the advertised memory minus the reserve. For example, with 13.5 GB of advertised memory and a 25% reserve, the `maxmemory` is about 10.1 GB.

> **Important**
>
> The `reserved-memory-percent` defaults to 25%. Do not reduce this value. A lower reserve can cause data loss, especially during upgrades or other changes to your cluster.

**Topics**


- How much memory to reserve
- Configuring reserved memory


## How much memory to reserve

The `reserved-memory-percent` parameter defaults to 25%. Do not reduce this value. However, there are situations where you can benefit from increasing the memory reserve.

Micro and small instances

The micro and small instances are sized for light activity without the burden of replication. To improve operational reliability of these instances for production workloads, increase the memory reserve to 30% on small instances and 50% on micro instances.

Forcing a forked bgsave or full sync

During a traditional forked bgsave or full sync, Valkey and Redis OSS memory usage can double (or more), depending on the traffic load and pattern. If `FreeableMemory` is insufficient to run

a traditional forked operation, Amazon ElastiCache automatically switches to a forkless save mechanism. This forkless mechanism uses much less memory, but can add client latencies — especially with very large key/value items. If these latencies are impactful for your use case, increase the memory reserve to ensure enough memory for a traditional forked save.

If instructed by AWS Support

Unusual traffic patterns or requirements can result in unique memory needs. Increase `reserved-memory-percent` if directed by AWS Support.

## Configuring reserved memory

The `reserved-memory-percent` parameter is specific to ElastiCache. It isn't part of the Valkey or Redis OSS open source code. To change this value, use a custom parameter group. You can't modify default parameter groups.

If 25% works for you, no changes are needed. To use a different value, follow these steps.

**To change the reserved-memory-percent value**

1. Run the following AWS CLI command to create a custom parameter group. Set the family to match your engine version. For more information about creating parameter groups, see Creating an ElastiCache parameter group.

```
aws elasticache create-cache-parameter-group \
   --cache-parameter-group-name my-parameter-group \
   --description "Custom reserved memory settings" \
   --cache-parameter-group-family redis7
```

   You need only one custom parameter group per engine family.

2. Set `reserved-memory-percent` to the value you want. For more information about modifying parameter groups, see Modifying an ElastiCache parameter group.

```
aws elasticache modify-cache-parameter-group \
   --cache-parameter-group-name my-parameter-group \
   --parameter-name-values "ParameterName=reserved-memory-percent,
 ParameterValue=30"
```

3. Apply the custom parameter group to your cluster or replication group.

To modify a cluster, run the following AWS CLI command. For more information about modifying clusters, see Modifying an ElastiCache cluster.

```
aws elasticache modify-cache-cluster \
   --cache-cluster-id my-cluster \
   --cache-parameter-group-name my-parameter-group \
   --apply-immediately
```

To modify a replication group, run the following AWS CLI command. For more information about modifying replication groups, see Modifying a replication group.

```
aws elasticache modify-replication-group \
   --replication-group-id my-replication-group \
   --cache-parameter-group-name my-parameter-group \
   --apply-immediately
```

# Best practices when working with Valkey and Redis OSS node-based clusters

Multi-AZ use, having sufficient memory, cluster resizing and minimizing downtime are all useful concepts to keep in mind when working with node-based clusters in Valkey or Redis OSS. We recommend you review and follow these best practices.

**Topics**


- Minimizing downtime with Multi-AZ
- Ensuring you have enough memory to make a Valkey or Redis OSS snapshot
- Online cluster resizing
- Minimizing downtime during maintenance


## Minimizing downtime with Multi-AZ

There are a number of instances where ElastiCache Valkey or Redis OSS may need to replace a primary node; these include certain types of planned maintenance and the unlikely event of a primary node or Availability Zone failure.

This replacement results in some downtime for the cluster, but if Multi-AZ is enabled, the downtime is minimized. The role of primary node will automatically fail over to one of the read replicas. There is no need to create and provision a new primary node, because ElastiCache will handle this transparently. This failover and replica promotion ensure that you can resume writing to the new primary as soon as promotion is complete.

See Minimizing downtime in ElastiCache by using Multi-AZ with Valkey and Redis OSS, to learn more about Multi-AZ and minimizing downtime.

## Ensuring you have enough memory to make a Valkey or Redis OSS snapshot

### Snapshots and synchronizations

Valkey and Redis OSS support snapshots and synchronizations using a forkless save process that allows you to allocate more of your memory to your application's use without incurring increased swap usage during synchronizations and saves.

When you work with ElastiCache for Redis OSS, Redis OSS calls a background write command in a number of cases:

- When creating a snapshot for a backup.
- When synchronizing replicas with the primary in a replication group.
- When enabling the append-only file feature (AOF) for Redis OSS.
- When promoting a replica to primary (which causes a primary/replica sync).

Whenever Redis OSS executes a background write process, you must have sufficient available memory to accommodate the process overhead. Failure to have sufficient memory available causes the process to fail. Because of this, it is important to choose a node instance type that has sufficient memory when creating your Redis OSS cluster.

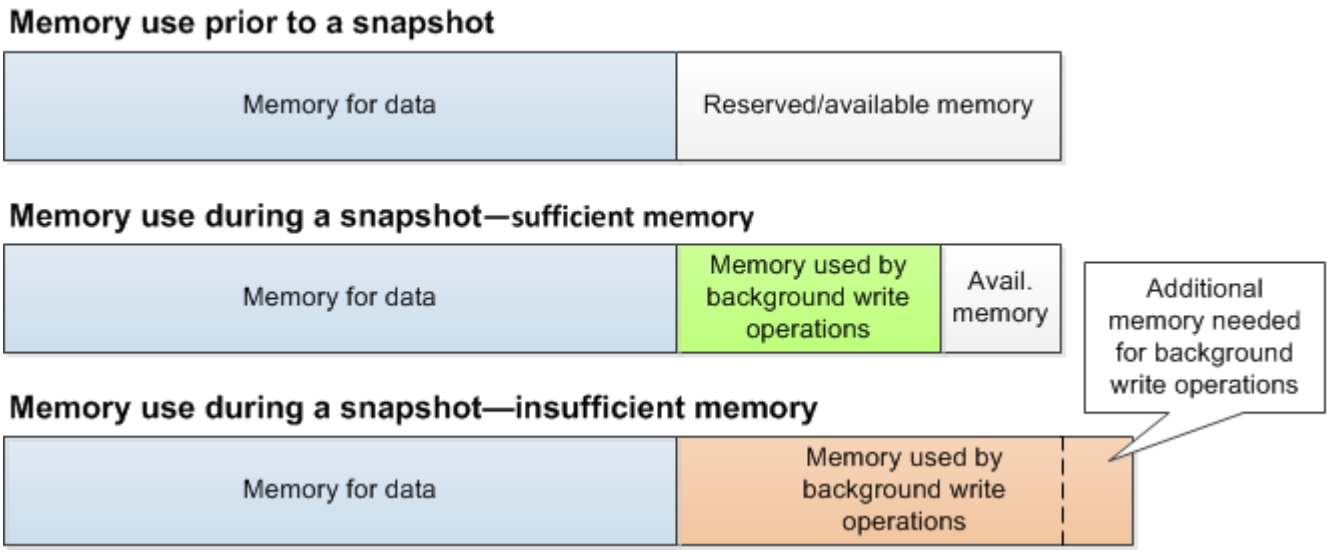
### Background Write Process and Memory Usage with Valkey and Redis OSS

Whenever a background write process is called, Valkey and Redis OSS forks its process (remember, these engines are single threaded). One fork persists your data to disk in a Redis OSS .rdb snapshot file. The other fork services all read and write operations. To ensure that your snapshot is a point-in-time snapshot, all data updates and additions are written to an area of available memory separate from the data area.

As long as you have sufficient memory available to record all write operations while the data is being persisted to disk, you should have no insufficient memory issues. You are likely to experience insufficient memory issues if any of the following are true:

- Your application performs many write operations, thus requiring a large amount of available memory to accept the new or updated data.
- You have very little memory available in which to write new or updated data.
- You have a large dataset that takes a long time to persist to disk, thus requiring a large number of write operations.

The following diagram illustrates memory use when executing a background write process.




For information on the impact of doing a backup on performance, see Performance impact of backups of node-based clusters.

For more information on how Valkey and Redis OSS perform snapshots, see http://valkey.io.

For more information on regions and Availability Zones, see Choosing regions and availability zones for ElastiCache.

**Avoiding running out of memory when executing a background write**

Whenever a background write process such as `BGSAVE` or `BGREWRITEAOF` is called, to keep the process from failing, you must have more memory available than will be consumed by write operations during the process. The worst-case scenario is that during the background write operation every record is updated and some new records are added to the cache. Because of this,

we recommend that you set `reserved-memory-percent` to 25 (25 percent) for Valkey and Redis OSS.

The `maxmemory` value indicates the memory available to you for data and operational overhead. Because you cannot modify the `reserved-memory` parameter in the default parameter group, you must create a custom parameter group for the cluster. The default value for `reserved-memory` is 0, which allows Redis OSS to consume all of *maxmemory* with data, potentially leaving too little memory for other uses, such as a background write process. For `maxmemory` values by node instance type, see Redis OSS node-type specific parameters.

You can also use `reserved-memory` parameter to reduce the amount of memory used on the box.

For more information on Valkey and Redis-specific parameters in ElastiCache, see Valkey and Redis OSS parameters.

For information on creating and modifying parameter groups, see Creating an ElastiCache parameter group and Modifying an ElastiCache parameter group.

## Online cluster resizing

*Resharding* involves adding and removing shards or nodes to your cluster and redistributing key spaces. As a result, multiple things have an impact on the resharding operation, such as the load on the cluster, memory utilization, and overall size of data. For the best experience, we recommend that you follow overall cluster best practices for uniform workload pattern distribution. In addition, we recommend taking the following steps.

Before initiating resharding, we recommend the following:

- **Test your application** – Test your application behavior during resharding in a staging environment if possible.
- **Get early notification for scaling issues** – Resharding is a compute-intensive operation. Because of this, we recommend keeping CPU utilization under 80 percent on multicore instances and less than 50 percent on single core instances during resharding. Monitor ElastiCache for Redis OSS metrics and initiate resharding before your application starts observing scaling issues. Useful metrics to track are `CPUUtilization`, `NetworkBytesIn`, `NetworkBytesOut`, `CurrConnections`, `NewConnections`, `FreeableMemory`, `SwapUsage`, and `BytesUsedForCacheItems`.

- **Ensure sufficient free memory is available before scaling in** – If you're scaling in, ensure that free memory available on the shards to be retained is at least 1.5 times the memory used on the shards you plan to remove.
- **Initiate resharding during off-peak hours** – This practice helps to reduce the latency and throughput impact on the client during the resharding operation. It also helps to complete resharding faster as more resources can be used for slot redistribution.
- **Review client timeout behavior** – Some clients might observe higher latency during online cluster resizing. Configuring your client library with a higher timeout can help by giving the system time to connect even under higher load conditions on server. In some cases, you might open a large number of connections to the server. In these cases, consider adding exponential backoff to reconnect logic. Doing this can help prevent a burst of new connections hitting the server at the same time.
- **Load your Functions on every shard** – When scaling out your cluster, ElastiCache will automatically replicate the Functions loaded in one of the existing nodes (selected at random) to the new node(s). If your cluster has Valkey 7.2 and above, or Redis OSS 7.0 or above, and your application uses Functions, we recommend loading all of your functions to all the shards before scaling out so that your cluster does not end up with different functions on different shards.

After resharding, note the following:

- Scale-in might be partially successful if insufficient memory is available on target shards. If such a result occurs, review available memory and retry the operation, if necessary. The data on the target shards will not be deleted.
- `FLUSHALL` and `FLUSHDB` commands are not supported inside Lua scripts during a resharding operation. Prior to Redis OSS 6, the `BRPOPLPUSH` command is not supported if it operates on the slot being migrated.

## Minimizing downtime during maintenance

Cluster mode configuration has the best availability during managed or unmanaged operations. We recommend that you use a cluster mode supported client that connects to the cluster discovery endpoint. For cluster mode disabled, we recommend that you use the primary endpoint for all write operations.

For read activity, applications can also connect to any node in the cluster. Unlike the primary endpoint, node endpoints resolve to specific endpoints. If you make a change in your cluster, such

as adding or deleting a replica, you must update the node endpoints in your application. This is why for cluster mode disabled, we recommend that you use the reader endpoint for read activity.

If AutoFailover is enabled in the cluster, the primary node might change. Therefore, the application should confirm the role of the node and update all the read endpoints. Doing this helps ensure that you aren't causing a major load on the primary. With AutoFailover disabled, the role of the node doesn't change. However, the downtime in managed or unmanaged operations is higher as compared to clusters with AutoFailover enabled.

Avoid directing read requests to a single read replica node, as its unavailability could lead to a read outage. Either fallback to reading from the primary, or ensure that you have at least two read replicas to avoid any read interruption during maintenance.

# Caching database query results

A common pattern to reduce database query latencies is query caching. Applications implement query caching by querying the cache for results associated with a database query, returning those results if they are cached. If no cached results are found, the query is executed on the database, the results are populated in the cache, and the results are then returned to the application initiating the query. Subsequent database queries will then return cached results as long as they remain in the cache.

## When to use query caching

Query caching with ElastiCache is most effective for the following workload types:

- **Read-heavy applications** where the same queries are executed repeatedly with data that changes infrequently.
- **Expensive queries** such as non-indexed lookups, aggregations, or multi-table joins where query execution times are long.
- **High-concurrency scenarios** where offloading reads to ElastiCache reduces database CPU pressure and improves overall throughput.

Query caching is not recommended for queries where strong consistency is required, or for queries inside multi-statement transactions that require read-after-write consistency.

## Using the AWS Advanced JDBC Wrapper

If your application is using a JDBC driver to query a relational database, you can implement query caching with the Remote Query Cache Plugin in the AWS Advanced JDBC Wrapper. The plugin automatically caches selected SQL query result sets in ElastiCache, returning the result set from the cache instead of the database for future queries. Caching query results can reduce database load and lower average query latencies with minimal application code changes.

## How query caching works with the plugin

The Remote Query Cache Plugin makes it easy for Java applications that query PostgreSQL, MySQL, or MariaDB databases to automatically cache query results in ElastiCache. You configure the plugin with your cache endpoint information and indicate which queries in your code to cache using query hints. When the plugin detects a hinted query, it returns the query result from the cache if present (a cache hit). If the results are not in the cache (a cache miss), the plugin

executes the query on the database, stores the results in the cache, and returns the results to your application so the next time the query is executed the results can be served from the cache.

### Caching hints

You control which queries to cache by setting hints on each query. You can apply query hints directly to query strings in your application code with a comment prefix:

```
/* CACHE_PARAM(ttl=300s) */ SELECT * FROM my_table WHERE id = 42
```

where `ttl` is the time-to-live in seconds. You can also set query hints in prepared statements using common frameworks like Hibernate and Spring Boot.

### Prerequisites

To use the Remote Query Cache Plugin in your application, you need an ElastiCache for Valkey or Redis OSS cache (both Serverless and node-based are supported) along with the following dependencies:

- AWS Advanced JDBC Wrapper version 3.3.0 or later.
- Apache Commons Pool version 2.11.1 or later.
- Valkey Glide version 2.3.0 or later.

### Example: Caching a query with the plugin

The following example shows how to enable the plugin and cache a query result for 300 seconds (5 minutes) with an ElastiCache for Valkey serverless cache:

```
import java.sql.*;
import java.util.Properties;

public class QueryCacheExample {
    public static void main(String[] args) throws SQLException {
        Properties props = new Properties();
        props.setProperty("user", "myuser");
        props.setProperty("password", "mypassword");

        // Enable the remote query cache plugin
        props.setProperty("wrapperPlugins", "remoteQueryCache");
```

```
        // Point to your ElastiCache endpoint
        props.setProperty("cacheEndpointAddrRw", "my-
cache.serverless.use1.cache.amazonaws.com:6379");

        Connection conn = DriverManager.getConnection(
            "jdbc:aws-wrapper:postgresql://my-database.cluster-abc123.us-
east-1.rds.amazonaws.com:5432/mydb",
            props
        );

        Statement stmt = conn.createStatement();

        // The SQL comment hint tells the plugin to cache this query for 300 seconds
        ResultSet rs = stmt.executeQuery(
            "/* CACHE_PARAM(ttl=300s) */ SELECT product_name, price FROM products WHERE
 category = 'electronics'"
        );

        while (rs.next()) {
            System.out.println(rs.getString("product_name") + ": $" +
 rs.getBigDecimal("price"));
        }

        rs.close();
        stmt.close();
        conn.close();
    }
}
```

The first time this query runs, the result is returned from the database and cached in ElastiCache. For the next 300 seconds, subsequent executions of that query are served directly from the cache.

### Related resources

You can find more extensive examples and detailed information about plugin configuration in the Remote Query Cache Plugin documentation.

## Caching strategies for Memcached

In the following topic, you can find strategies for populating and maintaining your Memcached cache.

What strategies to implement for populating and maintaining your cache depend upon what data you cache and the access patterns to that data. For example, you likely don't want to use the same strategy for both a top-10 leaderboard on a gaming site and trending news stories. In the rest of this section, we discuss common cache maintenance strategies and their advantages and disadvantages.

**Topics**


- Read replicas
- Lazy loading
- Write-through
- Adding TTL
- Related topics


## Read replicas

You can often significantly improve performance for ElastiCache serverless caches by reading from replicas instead of the primary cache node. For more information see Best Practices for using Read Replicas.

## Lazy loading

As the name implies, *lazy loading* is a caching strategy that loads data into the cache only when necessary. It works as described following.

Amazon ElastiCache is an in-memory key-value store that sits between your application and the data store (database) that it accesses. Whenever your application requests data, it first makes the request to the ElastiCache cache. If the data exists in the cache and is current, ElastiCache returns the data to your application. If the data doesn't exist in the cache or has expired, your application requests the data from your data store. Your data store then returns the data to your application. Your application next writes the data received from the store to the cache. This way, it can be more quickly retrieved the next time it's requested.

A *cache hit* occurs when data is in the cache and isn't expired:

1. Your application requests data from the cache.
2. The cache returns the data to the application.

A *cache miss* occurs when data isn't in the cache or is expired:

1. Your application requests data from the cache.
2. The cache doesn't have the requested data, so returns a `null`.
3. Your application requests and receives the data from the database.
4. Your application updates the cache with the new data.

**Advantages and disadvantages of lazy loading**

The advantages of lazy loading are as follows:

- Only requested data is cached.

  Because most data is never requested, lazy loading avoids filling up the cache with data that isn't requested.
- Node failures aren't fatal for your application.

  When a node fails and is replaced by a new, empty node, your application continues to function, though with increased latency. As requests are made to the new node, each cache miss results in a query of the database. At the same time, the data copy is added to the cache so that subsequent requests are retrieved from the cache.

The disadvantages of lazy loading are as follows:

- There is a cache miss penalty. Each cache miss results in three trips:
  1. Initial request for data from the cache
  2. Query of the database for the data
  3. Writing the data to the cache

  These misses can cause a noticeable delay in data getting to the application.
- Stale data.

  If data is written to the cache only when there is a cache miss, data in the cache can become stale. This result occurs because there are no updates to the cache when data is changed in the database. To address this issue, you can use the Write-through and Adding TTL strategies.

### Lazy loading pseudocode example

The following is a pseudocode example of lazy loading logic.

```
// ****************************************
// function that returns a customer's record.
// Attempts to retrieve the record from the cache.
// If it is retrieved, the record is returned to the application.
// If the record is not retrieved from the cache, it is
//    retrieved from the database,
//    added to the cache, and
//    returned to the application
// ****************************************
get_customer(customer_id)

    customer_record = cache.get(customer_id)
    if (customer_record == null)

        customer_record = db.query("SELECT * FROM Customers WHERE id = {0}",
 customer_id)
        cache.set(customer_id, customer_record)

    return customer_record
```

For this example, the application code that gets the data is the following.

```
customer_record = get_customer(12345)
```

## Write-through

The write-through strategy adds data or updates data in the cache whenever data is written to the database.

### Advantages and disadvantages of write-through

The advantages of write-through are as follows:

- Data in the cache is never stale.

  Because the data in the cache is updated every time it's written to the database, the data in the cache is always current.

- Write penalty vs. read penalty.

  Every write involves two trips:

  1. A write to the cache
  2. A write to the database

  Which adds latency to the process. That said, end users are generally more tolerant of latency when updating data than when retrieving data. There is an inherent sense that updates are more work and thus take longer.

The disadvantages of write-through are as follows:

- Missing data.

  If you spin up a new node, whether due to a node failure or scaling out, there is missing data. This data continues to be missing until it's added or updated on the database. You can minimize this by implementing lazy loading with write-through.

- Cache churn.

  Most data is never read, which is a waste of resources. By adding a time to live (TTL) value, you can minimize wasted space.

**Write-through pseudocode example**

The following is a pseudocode example of write-through logic.

```
// ****************************************
// function that saves a customer's record.
// ****************************************
save_customer(customer_id, values)

    customer_record = db.query("UPDATE Customers WHERE id = {0}", customer_id, values)
    cache.set(customer_id, customer_record)
    return success
```

For this example, the application code that gets the data is the following.

```
save_customer(12345,{"address":"123 Main"})
```

## Adding TTL

Lazy loading allows for stale data but doesn't fail with empty nodes. Write-through ensures that data is always fresh, but can fail with empty nodes and can populate the cache with superfluous data. By adding a time to live (TTL) value to each write, you can have the advantages of each strategy. At the same time, you can and largely avoid cluttering up the cache with extra data.

*Time to live (TTL)* is an integer value that specifies the number of seconds until the key expires. Valkey or Redis OSS can specify seconds or milliseconds for this value. Memcached specifies this value in seconds. When an application attempts to read an expired key, it is treated as though the key is not found. The database is queried for the key and the cache is updated. This approach doesn't guarantee that a value isn't stale. However, it keeps data from getting too stale and requires that values in the cache are occasionally refreshed from the database.

For more information, see the Valkey and Redis OSS commands or the Memcached `set` commands.

### TTL pseudocode examples

The following is a pseudocode example of write-through logic with TTL.

```
// ****************************************
// function that saves a customer's record.
// The TTL value of 300 means that the record expires
//    300 seconds (5 minutes) after the set command
//    and future reads will have to query the database.
// ****************************************
save_customer(customer_id, values)

    customer_record = db.query("UPDATE Customers WHERE id = {0}", customer_id, values)
    cache.set(customer_id, customer_record, 300)

    return success
```

The following is a pseudocode example of lazy loading logic with TTL.

```
// ****************************************
// function that returns a customer's record.
// Attempts to retrieve the record from the cache.
// If it is retrieved, the record is returned to the application.
// If the record is not retrieved from the cache, it is
//    retrieved from the database,
```

```
//    added to the cache, and
//    returned to the application.
// The TTL value of 300 means that the record expires
//    300 seconds (5 minutes) after the set command
//    and subsequent reads will have to query the database.
// ***************************************
get_customer(customer_id)

    customer_record = cache.get(customer_id)

    if (customer_record != null)
        if (customer_record.TTL < 300)
            return customer_record        // return the record and exit function

    // do this only if the record did not exist in the cache OR
    //    the TTL was >= 300, i.e., the record in the cache had expired.
    customer_record = db.query("SELECT * FROM Customers WHERE id = {0}", customer_id)
    cache.set(customer_id, customer_record, 300)  // update the cache
    return customer_record                // return the newly retrieved record and exit
 function
```

For this example, the application code that gets the data is the following.

```
save_customer(12345,{"address":"123 Main"})
```

```
customer_record = get_customer(12345)
```

### Related topics

- In-Memory Data Store
- Choosing an engine and version
- Scaling ElastiCache

## Amazon ElastiCache (Valkey) for e-commerce applications

An e-commerce application benefits from an in-memory cache layer between the application servers and the database. Amazon ElastiCache running Valkey provides sub-millisecond read latency for frequently accessed data — product catalog entries, inventory counts, search results, and user sessions — reducing database load and improving response times during traffic spikes.

## Cluster configuration

Choose a cluster type based on your data volume, throughput requirements, and operational preferences.

**Cluster type comparison for e-commerce**

| Cluster type | Best for | Scaling | Considerations |
|---|---|---|---|
| Serverless | Variable traffic patterns, new applications, teams without cache operations expertise | Automatic — scales compute and memory based on demand | No capacity planning needed. Variable cost model—you pay for what you consume. |
| Node-based (cluster mode enabled) | Predictable high-throughput workloads, need for fine-grained control over sharding and node types | Manual or auto-scaling of shards and replicas | Requires capacity planning. Fixed cost model—you pay for provisioned capacity regardless of utilization. Supports up to 500 nodes per cluster. |

For most e-commerce applications starting out, Serverless provides the simplest path to production. Migrate to node-based clusters when you have predictable traffic baselines and need more control over cluster topology, node types, and scaling behavior.

**Recommended cluster settings**

| Setting | Value | Rationale |
|---|---|---|
| Engine | Latest stable version | Use the latest stable Valkey release available in ElastiCache. Fully compatible with Redis OSS commands. Provides performance improvements over previous versions. |

| Setting | Value | Rationale |
| --- | --- | --- |
| Multi-AZ | Enabled | Automatic failover to a replica in another Availability Zone if the primary node fails. Required for production e-commerce workloads. |
| In-transit encryption | Enabled (TLS) | Encrypts data between your application and the cache cluster. Required for workloads handling user sessions or any PII. |
| At-rest encryption | Enabled | Encrypts data on disk (backups, swap). Required for compliance workloads. |
| Subnet group | Private isolated subnets (2+ AZs) | No internet access. Only reachable from your application's security group. |

## Key design for product data

Design cache keys to be predictable, debuggable, and scoped to avoid collisions across data types.

### Key naming conventions

| Data type | Key pattern | Value type | Example |
| --- | --- | --- | --- |
| Product details | `product:{id}` | Hash | `product:12345` → {name, price, description, imageUrl} |
| Inventory count | `inventory:{sku}` | String (integer) | `inventory:SKU-A100` → 42 |
| User session | `session:{sessionId}` | Hash | `session:abc123` → {userId, cart, lastAccess} |

| Data type | Key pattern | Value type | Example |
|---|---|---|---|
| Search results | `search:{queryHash}` | String (JSON) | `search:sha256(q=shoes&page=1)` → [product IDs] |
| Category listing | `category:{slug}:page:{n}` | List | `category:electronics:page:1` → [product IDs] |

Key design best practices:

- Use colons as separators for readability and tooling support.
- Keep keys short — long keys consume memory and network bandwidth.
- Include the data type prefix to avoid collisions between products, sessions, and other entities that might share numeric IDs.
- Use hashes for multi-field objects (products, sessions) to allow partial reads and updates without retrieving the entire value.

## TTL strategy by data type

Set TTL (time-to-live) values based on how frequently data changes and how stale it can be without affecting the customer experience.

**Recommended TTLs for e-commerce data**

| Data type | TTL | Invalidation trigger | Rationale |
|---|---|---|---|
| Product details | 5 minutes | Seller edits product | Product descriptions and images change infrequently. Short enough to pick up edits reasonably quickly without explicit invalidation for every change. |
| Inventory count | 30 seconds | Purchase or restock | Stale inventory can cause overselling. Very short TTL |

| Data type | TTL | Invalidation trigger | Rationale |
|---|---|---|---|
| | | | ensures counts refresh frequently. Explicit invalidat ion on purchase for immediate accuracy. |
| Search results | 60 seconds | None (TTL-based only) | Search indexes update periodically. Caching reduces search engine load. New products appear within 60 seconds without explicit invalidation. |
| User session | 24 hours | Logout or session expiry | Sessions persist across browsing. Refresh TTL on each access to keep active sessions alive. Delete explicitl y on logout. |
| Category listing | 2 minutes | Product added/rem oved from category | Category pages are high-traffic. Brief caching reduces database queries substantially during browsing. |

## Cache-aside pattern

The cache-aside pattern (also called lazy loading) is the most common caching strategy for e-commerce applications. Your application checks the cache first, and only queries the database on a cache miss.

### Read path (pseudocode)

```
FUNCTION getProduct(productId):
    cacheKey = "product:" + productId

    // Step 1: Check the cache
    cachedValue = cache.GET(cacheKey)
```

```
    IF cachedValue exists:
        RETURN deserialize(cachedValue)    // Cache hit — sub-millisecond

    // Step 2: Cache miss — query database
    product = database.query("SELECT * FROM products WHERE id = ?", productId)

    IF product not found:
        RETURN null

    // Step 3: Write to cache with TTL
    cache.SET(cacheKey, serialize(product), EXPIRE = 300)    // 5 minutes

    RETURN product
```

### Write path (pseudocode)

```
FUNCTION updateProduct(productId, updatedFields):
    // Step 1: Update the database (source of truth)
    database.update("UPDATE products SET ... WHERE id = ?", updatedFields, productId)

    // Step 2: Invalidate the cache (don't update it)
    cache.DELETE("product:" + productId)

    // Next read will trigger a cache miss and repopulate from database
```

**Important**

Always invalidate (delete) rather than update the cache on writes. This reduces the window for race conditions where a stale read overwrites a newer value. The next read repopulates the cache from the database, which is the source of truth. Note that a narrow race condition still exists—if a concurrent read fetches data from the database before the write occurs, it can repopulate the cache with stale data after the cache key has already been deleted. For most e-commerce workloads, the short TTL makes this acceptable. If you need strict consistency, use a distributed lock or versioned writes.

### Handling cache failures

```
FUNCTION getProductWithFallback(productId):
```

```
    TRY:
        RETURN getProduct(productId)    // Normal cache-aside path
    CATCH cacheConnectionError:
        // Cache is unavailable — fall back to database directly
        RETURN database.query("SELECT * FROM products WHERE id = ?", productId)
        // Log the error, trigger an alarm, but don't fail the request
```

Design your application so cache failure degrades performance (slower responses) but doesn't break functionality. The database serves as the fallback.

## Cache invalidation strategies

Invalidation ensures users see current data after updates. Choose a strategy based on how quickly changes must be visible.

### Invalidation approaches

| Approach | How it works | When to use |
|---|---|---|
| Delete on write | Application deletes the cache key immediately after updating the database. | Most writes (product edits, inventory changes). Simple, reliable, avoids stale data. |
| TTL expiration only | Don't invalidate — let the TTL expire naturally. | Data where brief staleness is acceptable (search results, category listings, analytics). |
| Event-driven invalidation | A background process listens to database change events and invalidates affected keys. | Systems where the write path and cache are in different services, or when a single database change affects many cache keys. |

For marketplace applications that also use Amazon CloudFront as a CDN layer, coordinate invalidation across both tiers: delete the ElastiCache key *and* send a CloudFront cache invalidation (or cache-tag invalidation) so users see updated content at both the application and edge layers.

## Connection management

- **Use connection pooling** — Creating a new TLS connection for each cache operation adds latency. Maintain a pool of persistent connections and reuse them across requests.
- **Set connection timeouts** — Use a short connection timeout (1–2 seconds) and a shorter command timeout (100–500 ms). If the cache doesn't respond quickly, fall back to the database rather than blocking the request.
- **Handle failover gracefully** — When Multi-AZ failover occurs, connections to the old primary break. Your connection pool should detect broken connections and reconnect automatically. Most client libraries handle this, but verify the behavior under test.
- **Use the cluster's configuration endpoint** — For cluster mode enabled, connect to the configuration endpoint rather than individual node endpoints. The configuration endpoint routes requests to the correct shard automatically.

## Key metrics to monitor

**Recommended Amazon CloudWatch alarms for e-commerce cache**

| Metric | Threshold | Period | Action |
|---|---|---|---|
| CacheHitRate | < 80% | 5 min | Investigate — low hit rate means your TTLs may be too short, keys are poorly designed, or working set exceeds memory. |
| EngineCPUUtilization | > 70% | 5 min | Scale up (larger nodes) or scale out (more shards). High CPU indicates the cache is processin g more commands than it can handle efficiently. |
| DatabaseMemoryUsag ePercentage | > 80% | 5 min | Risk of evictions. Increase memory (scale up) or reduce stored data (shorter TTLs, fewer cached data types). |
| Evictions | > 0 sustained | 1 min | Cache is full and removing data to make room. Increases cache |

| Metric | Threshold | Period | Action |
| --- | --- | --- | --- |
| | | | misses. Scale up memory or reduce TTLs on less-critical data. |
| CurrConnections | > 80% of your client pool size | 5 min | Connection pool nearing exhaustion. Increase pool size, reduce connection hold time, or investigate connection leaks in application code. Base the threshold on your application's configured pool size, not the server maximum (65,000). |

## Frequently asked questions

**When should I use Serverless vs. node-based?**

Use Serverless when your traffic is unpredictable (new marketplace, seasonal spikes), when you want to avoid capacity planning, or when your team doesn't have caching operations expertise. Switch to node-based when your traffic patterns are stable, you need fine-grained control over sharding, or your sustained throughput makes node-based more cost-effective.

**Should I use Valkey or Redis OSS?**

Use Valkey for new deployments. Valkey is the default engine for new ElastiCache clusters, is fully compatible with Redis OSS commands and data structures, and receives continued development. Existing Redis OSS clusters continue to work — migrate to Valkey when convenient using the in-place engine upgrade.

**What happens when the cache is unavailable?**

Your application should treat the cache as an optimization, not a dependency. If the cache is unavailable, fall back to querying the database directly. Response times will be higher, but the application remains functional. With Multi-AZ enabled, complete cache unavailability is rare — failover to a replica typically completes in under 30 seconds.

**How do I estimate the memory I need?**

Calculate: (number of unique items to cache) × (average size per item) × (overhead factor of 1.2 for Valkey data structures). The overhead varies by object size and data type—smaller objects have proportionally higher overhead. For example, 100,000 products at 2 KB each with 1.2x overhead = approximately 240 MB. Add session data, search results, and inventory counts. Start with headroom (use 60% of available memory as your target) and monitor the DatabaseMemoryUsagePercentage metric to adjust.

# Durability in ElastiCache

Amazon ElastiCache for Valkey now supports durability. This capability enables you to use ElastiCache for both caching and data store workloads with microsecond read latency. Durability provides data protection through a Multi-AZ transactional log, enabling fast recovery and restart during failures. ElastiCache offers two durability options: synchronous writes designed for zero data loss at single-digit millisecond write latency, and asynchronous writes for microsecond write latency. Durability is currently supported with node-based ElastiCache.

When configuring an ElastiCache cluster, you can choose between two durability options. With synchronous writes, data is persisted across at least two Availability Zones (AZs) in a Multi-AZ transactional log before responding to the client. While this provides data durability during failures, it increases write latency from microseconds to single-digit milliseconds. With asynchronous writes, data is persisted in a Multi-AZ transactional log after responding to the client, maintaining ElastiCache's microsecond write latency with the risk of losing up to 10 seconds of uncommitted data during a failure. Both options maintain microsecond read latencies. With asynchronous writes, if the primary node cannot persist changes to the Multi-AZ transactional log for more than 10 seconds, it will reject incoming write commands until the ability to persist writes is restored. If your application cannot tolerate any data loss, we recommend synchronous writes. Otherwise, we recommend asynchronous writes. The durability options give you flexibility to balance between write performance and data durability based on your specific workload requirements.

**Topics**


- Durability options
- Configuring durability
- Changing durability on an existing cluster
- Consistency

- Failure scenarios
- Limitations
- Monitoring
- Getting started with durability

## Durability options

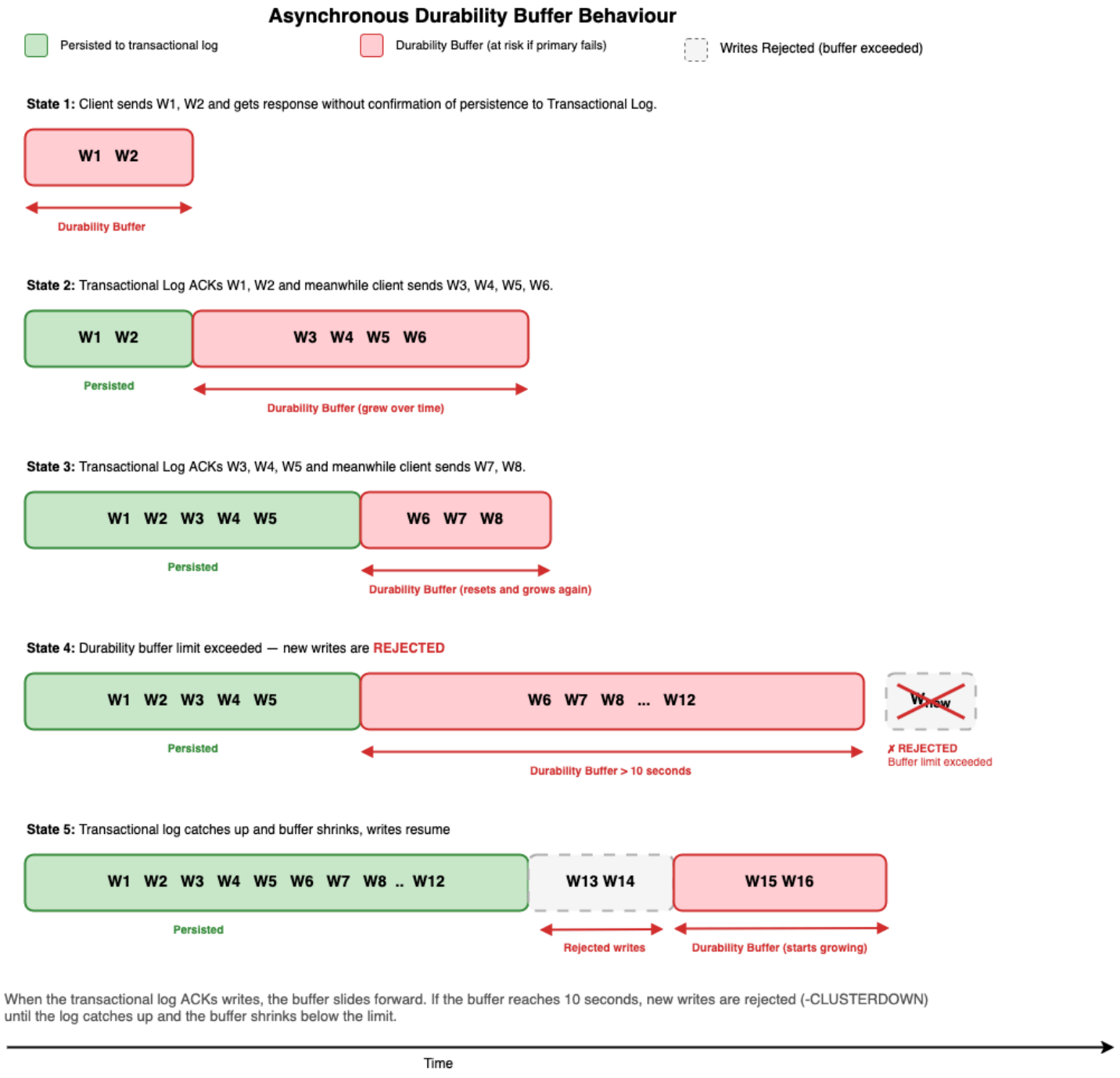
ElastiCache for Valkey offers two durability options: synchronous and asynchronous writes.

With synchronous writes, successful write operations are durably stored in the Multi-AZ transactional log before returning to clients. This incurs single-digit millisecond write latency and ensures that no acknowledged write operations will be lost in the event of a failure.

With asynchronous writes, successful write operations are returned to clients before being durably stored in the Multi-AZ transactional log. Since the write operations don't wait to be durably stored in the Multi-AZ transactional log, the write operation latency is equivalent to ElastiCache without durability. However, up to the last 10 seconds of successful write operations may be lost in the event of a failure.

To understand the potential data loss with asynchronous writes, consider the concept of a durability buffer. The durability buffer represents the maximum age of any write that has been accepted by the primary node but not yet persisted to the Multi-AZ transactional log. The primary node tracks the age of the oldest unacknowledged write. As long as this age remains under 10 seconds, the node continues accepting new writes normally. If the age of the oldest unacknowledged write grows beyond 10 seconds, the primary node will reject all incoming write commands until it catches up. Read operations continue to be served at microsecond latency during this period. Once the pending writes are persisted, the node resumes accepting writes automatically. This ensures that potential data loss is limited to 10 seconds worth of writes in the event of a failure.

When configuring your client to send traffic to an asynchronous durable cluster, ensure that the client automatically retries with exponential backoff any write commands that are rejected with the cluster down error message. For guidance on configuring your clients to handle this and other transient errors, see Best practices: Valkey/Redis OSS clients and Amazon ElastiCache.




## Choosing a durability option

Use synchronous writes when your application cannot tolerate any data loss during failures. With synchronous writes, you can use ElastiCache for a broader set of use cases beyond caching where data loss is not acceptable, such as knowledge bases for RAG applications, AI agent memory, AI agent workflow state, payment tokenization, streaming metadata, gaming player state, and real-time inventory management.

Use asynchronous writes when your application prioritizes write performance and can tolerate the potential loss of up to 10 seconds of uncommitted data during a failure. This option is ideal for workloads such as application data caching, session stores, gaming leaderboards, and real-time analytics.

> ⚠ **Eviction policy consideration for durable clusters**
>
> Durable clusters use the same default parameter group as non-durable clusters. This parameter group sets `maxmemory-policy` to `volatile-lru`. With this policy, Amazon ElastiCache might evict keys that have a time to live (TTL) set when memory is under pressure. This can happen even on a durable cluster.
>
> To prevent eviction of keys with a TTL, create a custom parameter group and set `maxmemory-policy` to `noeviction`. With `noeviction`, write commands return an error when memory is full rather than removing keys.
>
> Monitor `BytesUsedForCache` and `DatabaseMemoryUsagePercentage` to make sure your cluster has enough memory for your workload.

## Configuring durability

When creating an ElastiCache cluster, you can set the `--durability` parameter to one of the following values:

**Durability parameter values**

| --durability value | Behavior | EffectiveDurability (Valkey 9.0) |
|---|---|---|
| `sync` | Writes persisted to Multi-AZ transactional log before responding to client | sync |
| `async` | Writes persisted to Multi-AZ transactional log after responding to client | async |
| `disabled` | No durability; standard ElastiCache behavior | disabled |
| `default` | ElastiCache determines setting based on engine version and cluster configuration | disabled |

The `EffectiveDurability` property in the API response always shows the actual durability mode in effect (sync, async, or disabled). You can switch between the sync, async, and disabled modes on an existing cluster by calling `ModifyReplicationGroup` with the `--durability` parameter. For more information, see Changing durability on an existing cluster.

For snapshots, ElastiCache stores the `EffectiveDurability` value so that restoring from a snapshot preserves the actual durability setting.

When you enable durability, ElastiCache automatically enables encryption at-rest on your cluster. All data at rest is encrypted, including snapshots and data in the Multi-AZ transactional log. By default, the cluster uses a service-managed KMS key, but you can also specify your own customer-managed KMS key. The encryption status is visible in the `StorageEncryptionType` field for the replication group.

## Changing durability on an existing cluster

You can switch an existing ElastiCache for Valkey cluster between the `sync`, `async`, and `disabled` durability modes without recreating it. ElastiCache applies the change to the live cluster while it continues to serve traffic. For information about tracking the change, see Monitor the durability change.

You can change the durability mode by using the AWS Management Console, the AWS CLI, or an AWS SDK. The following sections describe the console and AWS CLI methods. In the AWS CLI, you use `ModifyReplicationGroup` with the `--durability` parameter. In the ElastiCache API, the corresponding request member is `Durability`.

Before you change durability on an existing cluster, make sure that your cluster meets the following requirements:

- The cluster runs Valkey 9.0 or later.
- Multi-AZ is enabled with at least one replica per shard.
- The cluster has encryption in transit (TLS) enabled.
- The cluster uses Cluster Mode Enabled (CME).
- The cluster is not an ElastiCache Serverless cache.
- The cluster is not part of a Global Datastore.

When a cluster uses the `sync` or `async` mode, ElastiCache automatically enables encryption at rest on the cluster. For more information, see Configuring durability.

For the complete list of durability requirements and limitations, see Limitations.

Change the durability mode as a standalone modification. ElastiCache rejects a `ModifyReplicationGroup` request that changes durability together with any other modification.

> **Considerations**
>
> Changing the durability mode involves node replacement. For each shard, this causes a brief primary node change or an automatic failover. Clients can experience transient connection interruptions during this process. Clients might also need to refresh their view of the cluster topology. Configure your clients to reconnect and to retry write operations with exponential backoff. For guidance, see Best practices: Valkey/Redis OSS clients and Amazon ElastiCache.
> We recommend changing the durability mode during a planned operational window.

## Using the AWS Management Console

You can change the durability mode for an existing cluster from the ElastiCache console. On the cluster's **Modify** page (**Update cluster**), change the durability setting and save your changes.

## Using the AWS CLI

To change the durability mode, set the `--durability` parameter to `sync`, `async`, or `disabled`. Use `sync` when your application cannot tolerate any data loss. Use `async` when your application prioritizes write performance. With `async`, you can lose up to 10 seconds of acknowledged writes during a failure. If your application cannot tolerate that loss, use `sync` instead.

### Set the sync or async mode

The following AWS CLI command sets synchronous writes:

```
aws elasticache modify-replication-group \
  --replication-group-id my-valkey-cluster \
  --durability sync \
  --region us-east-1
```

To use asynchronous writes, set `--durability` to `async`.

When a cluster changes from `disabled` to `sync` or `async`, ElastiCache automatically enables encryption at rest, including snapshots and data in the Multi-AZ transactional log. If the cluster

already uses a customer managed AWS KMS key, ElastiCache retains that key. If the cluster was not previously encrypted, you don't need to specify a KMS key in this request. Changing the durability mode to `disabled` later does not disable encryption at rest.

### Set the disabled mode

If your workload no longer requires Multi-AZ transactional-log protection, you can set durability to `disabled`.

> **Data-loss risk when you set the disabled mode**
>
> Setting durability to `disabled` removes the Multi-AZ transactional-log durability guarantee. It increases the risk of losing acknowledged writes during a failure. Make sure you understand this recovery and data-loss tradeoff before you proceed.

The following AWS CLI command sets the disabled mode:

```
aws elasticache modify-replication-group \
  --replication-group-id my-durable-cluster \
  --durability disabled \
  --region us-east-1
```

## Monitor the durability change

The following AWS CLI command checks the status of a durability change:

```
aws elasticache describe-replication-groups \
  --replication-group-id my-durable-cluster \
  --query "ReplicationGroups[0].
{Id:ReplicationGroupId,Status:Status,Durability:Durability,EffectiveDurability:EffectiveDurabil
 \
  --region us-east-1
```

To block until the change finishes, use the `replication-group-available` waiter. The waiter polls until the cluster `Status` is `available`, which means the modification finished running.

```
aws elasticache wait replication-group-available \
  --replication-group-id my-durable-cluster \
  --region us-east-1
```

A status of `available` does not by itself confirm that the change succeeded, because a failed change can roll back. After the cluster is available, confirm that `EffectiveDurability` matches the durability mode you requested. If it does not match, the change might have rolled back; check the cluster's ElastiCache events for the reason. A requested `Durability` value of `default` can legitimately resolve to a different `EffectiveDurability` value.

For information about the CloudWatch metrics available for durable clusters, including `DurabilityLag` and `DurabilityBufferExceededErrorCount`, see Monitoring.

## Consistency

### Synchronous writes

Primary nodes are strongly consistent. Successful write operations are durably stored in the Multi-AZ transactional log before returning to clients. During normal operations, read operations on primaries always return the most up-to-date data reflecting the effects from all prior successful write operations. Such strong consistency is preserved across primary failovers. Replica nodes are eventually consistent. Read operations from replicas (using `READONLY` command) might not always reflect the effects of the most recent successful write operations, with lag metrics published to CloudWatch. The read operations from a single replica are sequentially consistent. Successful write operations take effect on each replica in the same order they were executed on the primary.

### Asynchronous writes

During normal operation, asynchronous writes provide the same consistency behavior as synchronous writes. However, as the write operations are returned to clients before being durably stored in the Multi-AZ transactional log, strong consistency is not preserved across primary failovers. In the event of a failure, up to 10 seconds of acknowledged write operations may be lost, meaning that after a failover, read operations on the new primary may not reflect all previously acknowledged writes. Asynchronous writes is not recommended for workloads that require strong consistency guarantees.

## Failure scenarios

If a primary node fails, ElastiCache automatically triggers a failover to a replica, and the new primary takes over write operations. The failed primary is replaced and syncs data from the Multi-AZ transactional log. With synchronous writes, this process ensures no data loss. With asynchronous writes, up to 10 seconds of uncommitted data can be lost during the failover. If a

read replica fails, the failed node is replaced and syncs data from the Multi-AZ transactional log regardless of the durability option selected.

When all nodes in a shard fail, all nodes are replaced and sync data from the Multi-AZ transactional log at the same time. With synchronous writes, this process ensures no data loss. With asynchronous writes, up to 10 seconds of uncommitted data can be lost. After committed data is restored, one of the replaced nodes will automatically be elected as the new primary with other nodes as replicas.

## Limitations

- Durability for ElastiCache is supported on Valkey 9.0 or later.
- Durability for ElastiCache is supported for the following instance type families: R8g, R7g, R6g, M8g, M7g, M6g, C8gn, and C7gn.
- Durability is not supported with ElastiCache Serverless.
- Durability is not supported with Global Datastores, Outposts, Local Zones, or data tiering.
- Durability enabled clusters support up to 100 MBps of write throughput per primary node.
- Durability for ElastiCache is not supported for Cluster Mode Disabled (CMD) clusters.
- Durability for ElastiCache requires Multi-AZ enabled with at least one replica per shard.
- Durability requires and automatically enables encryption at-rest.
- Changing durability on an existing cluster requires encryption in transit (TLS) to be enabled.
- Online migration from self-hosted Valkey or Redis OSS to a durable cluster is not supported.
- When durability is enabled and search indexes are configured, write commands targeting indexed keys may be throttled to maintain transactional log performance. For details, see Search write throttling.

## Monitoring

ElastiCache publishes the following additional CloudWatch metrics for durable clusters:

- **DurabilityLag**: The age in milliseconds of the oldest write operation that has been acknowledged to the client but not yet persisted to the Multi-AZ transactional log. When this value approaches 10,000 ms (10 seconds), the primary node will begin rejecting new writes. This metric is always 0 for synchronous writes because writes are persisted before being acknowledged.

- **DurabilityBufferExceededErrorCount**: The number of write commands rejected by a node because the durability buffer exceeded 10 seconds. When this metric is non-zero, your write requests will be rejected. Ensure your client is configured to retry with exponential backoff. This metric is always 0 for synchronous writes.
- **MaxReplicationThroughput**: The maximum observed replication throughput, in bytes per second. ElastiCache samples throughput over short intervals to catch traffic bursts, and reports the highest sample once per minute. For example, if your application writes 1 MB in 10 ms, this metric reports 100 MBps. Above 100 MBps, ElastiCache throttles write throughput, and you can see higher write latency. When this value approaches 100 MBps, add shards to spread write traffic across more primary nodes.

In addition to durability-specific metrics, ElastiCache also monitors write throughput. In the event that the incoming write traffic exceeds the available throughput of the Multi-AZ transactional log, incoming write commands will be throttled. In the event that customer traffic is throttled, the `TrafficManagementActive` metric will emit 1.

## Getting started with durability

You can create a durable ElastiCache cluster using the AWS Management Console or the AWS CLI.

### Using the AWS Management Console

To get started, create a new ElastiCache cluster with durability by selecting Valkey 9.0 or later in your desired AWS Region. After enabling durability, you can choose either synchronous or asynchronous options depending on your application requirements by using the AWS Console.

For more information on creating a cluster, see Creating a cluster for Valkey or Redis OSS.

### Using the CLI

To create a new durable cluster with synchronous writes:

```
aws elasticache create-replication-group \
  --replication-group-id my-durable-cluster \
  --replication-group-description "ElastiCache durable cluster" \
  --engine valkey --engine-version 9.0 \
  --num-node-groups 2 --replicas-per-node-group 1 \
  --cache-node-type cache.r7g.large \
  --multi-az-enabled \
  --transit-encryption-enabled \
```

```
  --durability sync \
  --region us-east-1
```

To create a cluster with asynchronous writes, set `--durability async`:

```
aws elasticache create-replication-group \
  --replication-group-id my-durable-cluster \
  --replication-group-description "ElastiCache durable cluster" \
  --engine valkey --engine-version 9.0 \
  --num-node-groups 2 --replicas-per-node-group 1 \
  --cache-node-type cache.r7g.large \
  --multi-az-enabled \
  --transit-encryption-enabled \
  --durability async \
  --region us-east-1
```

To switch an existing cluster between synchronous and asynchronous writes, use `modify-replication-group`:

```
aws elasticache modify-replication-group \
  --replication-group-id my-durable-cluster \
  --durability async
```

# Managing your node-based cluster in ElastiCache

ElastiCache offers two deployment options, serverless caches and node-based clusters. Each have their own capabilities and requirements.

This section contains topics to help you manage your node-based clusters.

> **Note**
>
> These topics do not apply to ElastiCache Serverless.

**Topics**


- Auto Scaling Valkey and Redis OSS clusters
- Modifying cluster mode
- Replication across AWS Regions using global datastores

- High availability using replication groups
- Managing ElastiCache cluster maintenance
- Configuring engine parameters using ElastiCache parameter groups

# Auto Scaling Valkey and Redis OSS clusters

## Prerequisites

ElastiCache Auto Scaling is limited to the following:

- Valkey or Redis OSS (cluster mode enabled) clusters running Valkey 7.2 onwards, or running Redis OSS 6.0 onwards
- Data tiering (cluster mode enabled) clusters running Valkey 7.2 onwards, or running Redis OSS 7.0.7 onwards
- Instance sizes - Large, XLarge, 2XLarge
- Instance type families - R8g, R8gd, R7g, R7gd, R6g, R6gd, R5, M8g, M7g, M6g, M5, C8gn, C7gn
- Auto Scaling in ElastiCache is not supported for clusters running in Global datastores, Outposts or Local Zones.

## Managing Capacity Automatically with ElastiCache Auto Scaling with Valkey or Redis OSS

ElastiCache auto scaling with Valkey or Redis OSS is the ability to increase or decrease the desired shards or replicas in your ElastiCache service automatically. ElastiCache leverages the Application Auto Scaling service to provide this functionality. For more information, see Application Auto Scaling. To use automatic scaling, you define and apply a scaling policy that uses CloudWatch metrics and target values that you assign. ElastiCache auto scaling uses the policy to increase or decrease the number of instances in response to actual workloads.

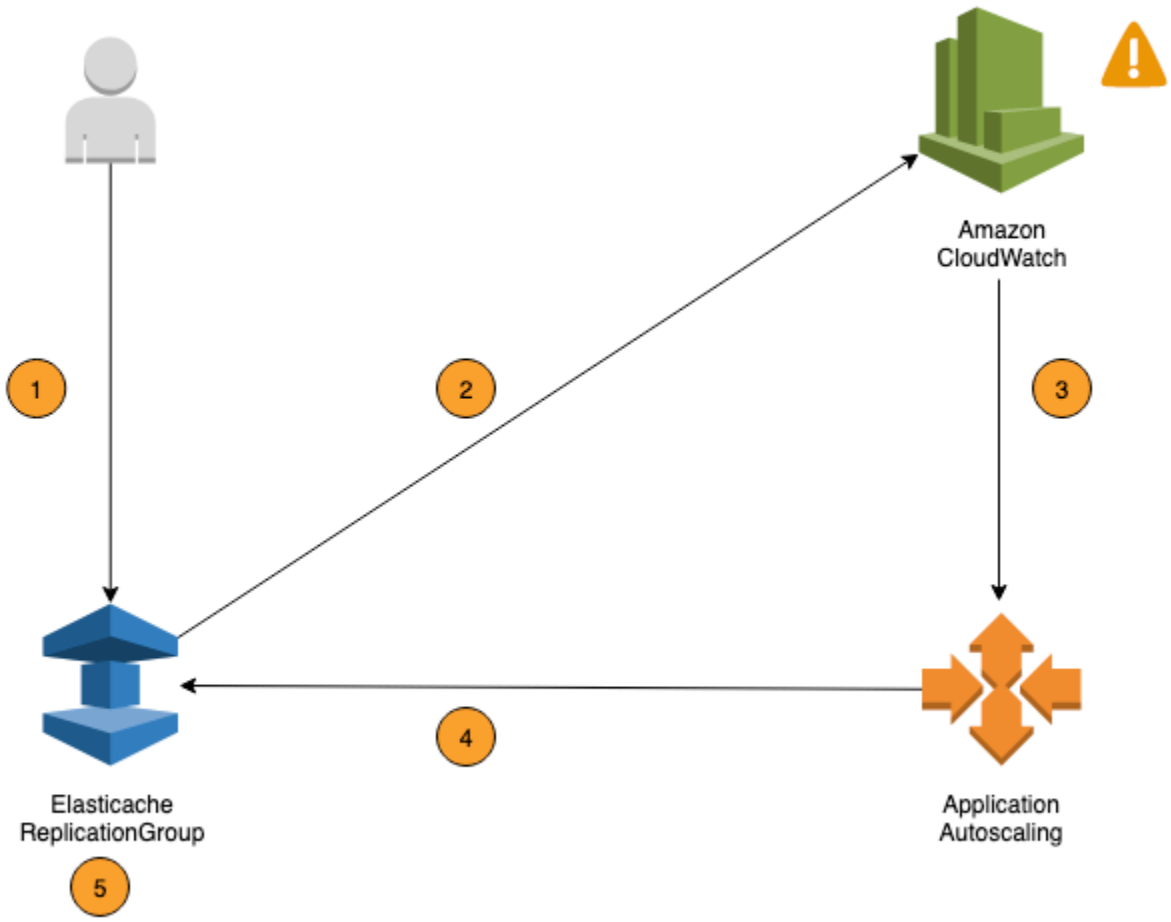
You can use the AWS Management Console to apply a scaling policy based on a predefined metric. A `predefined metric` is defined in an enumeration so that you can specify it by name in code or use it in the AWS Management Console. Custom metrics are not available for selection using the AWS Management Console. Alternatively, you can use either the AWS CLI or the Application Auto Scaling API to apply a scaling policy based on a predefined or custom metric.

ElastiCache for Valkey and Redis OSS supports scaling for the following dimensions:

- **Shards** – Automatically add/remove shards in the cluster similar to manual online resharding. In this case, ElastiCache auto scaling triggers scaling on your behalf.
- **Replicas** – Automatically add/remove replicas in the cluster similar to manual Increase/Decrease replica operations. ElastiCache auto scaling for Valkey and Redis OSS adds/removes replicas uniformly across all shards in the cluster.

ElastiCache for Valkey and Redis OSS supports the following types of automatic scaling policies:

- Target tracking scaling policies – Increase or decrease the number of shards/replicas that your service runs based on a target value for a specific metric. This is similar to the way that your thermostat maintains the temperature of your home. You select a temperature and the thermostat does the rest.
- Scheduled scaling for your application. – ElastiCache for Valkey and Redis OSS auto scaling can increase or decrease the number of shards/replicas that your service runs based on the date and time.




The following steps summarize the ElastiCache for Valkey and Redis OSS auto scaling process as shown in the previous diagram:

1. You create an ElastiCache auto scaling policy for your Replication Group.
2. ElastiCache auto scaling creates a pair of CloudWatch alarms on your behalf. Each pair represents your upper and lower boundaries for metrics. These CloudWatch alarms are triggered when the cluster's actual utilization deviates from your target utilization for a sustained period of time. You can view the alarms in the console.
3. If the configured metric value exceeds your target utilization (or falls below the target) for a specific length of time, CloudWatch triggers an alarm that invokes auto scaling to evaluate your scaling policy.
4. ElastiCache auto scaling issues a Modify request to adjust your cluster capacity.

5. ElastiCache processes the Modify request, dynamically increasing (or decreasing) the cluster Shards/Replicas capacity so that it approaches your target utilization.

To understand how ElastiCache Auto Scaling works, suppose that you have a cluster named `UsersCluster`. By monitoring the CloudWatch metrics for `UsersCluster`, you determine the Max shards that the cluster requires when traffic is at its peak and Min Shards when traffic is at its lowest point. You also decide a target value for CPU utilization for the `UsersCluster` cluster. ElastiCache auto scaling uses its target tracking algorithm to ensure that the provisioned shards of `UsersCluster` is adjusted as required so that utilization remains at or near to the target value.

> **Note**
>
> Scaling may take noticeable time and will require extra cluster resources for shards to rebalance. ElastiCache Auto Scaling modifies resource settings only when the actual workload stays elevated (or depressed) for a sustained period of several minutes. The auto scaling target-tracking algorithm seeks to keep the target utilization at or near your chosen value over the long term.

## Auto Scaling policies

A scaling policy has the following components:

- A target metric – The CloudWatch metric that ElastiCache for Valkey and Redis OSS Auto Scaling uses to determine when and how much to scale.
- Minimum and maximum capacity – The minimum and maximum number of shards or replicas to use for scaling.

> **Important**
>
> While creating Auto scaling policy , if current capacity is higher than max capacity configured, we scaleIn to the MaxCapacity during policy creation. Similarly if current capacity is lower than min capacity configured, we scaleOut to the MinCapacity.

- A cooldown period – The amount of time, in seconds, after a scale-in or scale-out activity completes before another scale-out activity can start.

- A service-linked role – An AWS Identity and Access Management (IAM) role that is linked to a specific AWS service. A service-linked role includes all of the permissions that the service requires to call other AWS services on your behalf. ElastiCache Auto Scaling automatically generates this role, `AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG`, for you.
- Enable or disable scale-in activities - Ability to enable or disable scale-in activities for a policy.

**Topics**


- Target metric for Auto Scaling
- Minimum and maximum capacity
- Cool down period
- Enable or disable scale-in activities


### Target metric for Auto Scaling

In this type of policy, a predefined or custom metric and a target value for the metric is specified in a target-tracking scaling policy configuration. ElastiCache for Valkey and Redis OSS Auto Scaling creates and manages CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and target value. The scaling policy adds or removes shards/replicas as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target-tracking scaling policy also adjusts to fluctuations in the metric due to a changing workload. Such a policy also minimizes rapid fluctuations in the number of available shards/replicas for your cluster.

For example, consider a scaling policy that uses the predefined average `ElastiCachePrimaryEngineCPUUtilization` metric. Such a policy can keep CPU utilization at, or close to, a specified percentage of utilization, such as 70 percent.

> **Note**
>
> For each cluster, you can create only one Auto Scaling policy for each target metric.

### Minimum and maximum capacity

#### Shards

You can specify the maximum number of shards that can be scaled to by ElastiCache for Valkey and Redis OSS auto scaling. This value must be less than or equal to 250 with a minimum of 1. You can also specify the minimum number of shards to be managed by auto scaling. This value must be at least 1, and equal to or less than the value specified for the maximum shards 250.

**Replicas**

You can specify the maximum number of replicas to be managed by ElastiCache for Valkey and Redis OSS auto scaling. This value must be less than or equal to 5. You can also specify the minimum number of replicas to be managed by auto scaling. This value must be at least 1, and equal to or less than the value specified for the maximum replicas 5.

To determine the minimum and maximum number of shards/replicas that you need for typical traffic, test your Auto Scaling configuration with the expected rate of traffic to your model.

> **Note**
>
> ElastiCache auto scaling policies increase cluster capacity until it reaches your defined maximum size or until service limits apply. To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

> **Important**
>
> Scaling-in occurs when there is no traffic. If a variant's traffic becomes zero, ElastiCache automatically scales in to the minimum number of instances specified.

**Cool down period**

You can tune the responsiveness of a target-tracking scaling policy by adding cooldown periods that affect scaling your cluster. A cooldown period blocks subsequent scale-in or scale-out requests until the period expires. This slows the deletions of shards/replicas in your ElastiCache for Valkey and Redis OSS cluster for scale-in requests, and the creation of shards/replicas for scale-out requests. You can specify the following cooldown periods:

- A scale-in activity reduces the number of shards/replicas in your cluster. A scale-in cooldown period specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in activity can start.

- A scale-out activity increases the number of shards/replicas in your cluster. A scale-out cooldown period specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start.

When a scale-in or a scale-out cooldown period is not specified, the default for scale-out is 600 seconds and for scale-in 900 seconds.

**Enable or disable scale-in activities**

You can enable or disable scale-in activities for a policy. Enabling scale-in activities allows the scaling policy to delete shards/replicas. When scale-in activities are enabled, the scale-in cooldown period in the scaling policy applies to scale-in activities. Disabling scale-in activities prevents the scaling policy from deleting shards/replicas.

> **Note**
>
> Scale-out activities are always enabled so that the scaling policy can create ElastiCache shards or replicas as needed.

## IAM Permissions Required for Auto Scaling

ElastiCache for Valkey and Redis OSS Auto Scaling is made possible by a combination of the ElastiCache, CloudWatch, and Application Auto Scaling APIs. Clusters are created and updated with ElastiCache, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling. In addition to the standard IAM permissions for creating and updating clusters, the IAM user that accesses ElastiCache Auto Scaling settings must have the appropriate permissions for the services that support dynamic scaling. In this most recent policy we have added support for Memcached vertical scaling, with the action `elasticache:ModifyCacheCluster`. IAM users must have permissions to use the actions shown in the following example policy:

JSON

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

```
            "Action": [
                "application-autoscaling:*",
                "elasticache:DescribeReplicationGroups",
                "elasticache:ModifyReplicationGroupShardConfiguration",
                "elasticache:IncreaseReplicaCount",
                "elasticache:DecreaseReplicaCount",
                "elasticache:DescribeCacheClusters",
                "elasticache:DescribeCacheParameters",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DisableAlarmActions",
                "cloudwatch:EnableAlarmActions",
                "iam:CreateServiceLinkedRole",
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:Get*",
                "sns:List*"
            ],
            "Resource": "arn:aws:iam::123456789012:role/autoscaling-roles-for-
cluster"
        }
    ]
}
```

## Service-linked role

The ElastiCache for Valkey and Redis OSS auto scaling service also needs permission to describe your clusters and CloudWatch alarms, and permissions to modify your ElastiCache target capacity on your behalf. If you enable Auto Scaling for your cluster, it creates a service-linked role named `AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG`. This service-linked role grants ElastiCache auto scaling permission to describe the alarms for your policies, to monitor the current capacity of the fleet, and to modify the capacity of the fleet. The service-linked role is the default role for ElastiCache auto scaling. For more information, see Service-linked roles for ElastiCache for Redis OSS auto scaling in the Application Auto Scaling User Guide.

## Auto Scaling Best Practices

Before registering to Auto Scaling, we recommend the following:

1. **Use just one tracking metric** – Identify if your cluster has CPU or data intensive workloads and use a corresponding predefined metric to define Scaling Policy.
   - Engine CPU: `ElastiCachePrimaryEngineCPUUtilization` (shard dimension) or `ElastiCacheReplicaEngineCPUUtilization` (replica dimension)
   - Database usage: `ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage` This scaling policy works best with maxmemory-policy set to noeviction on the cluster.

   We recommend you avoid multiple policies per dimension on the cluster. ElastiCache for Valkey and Redis OSS Auto scaling will scale out the scalable target if any target tracking policies are ready for scale out, but will scale in only if all target tracking policies (with the scale-in portion enabled) are ready to scale in. If multiple policies instruct the scalable target to scale out or in at the same time, it scales based on the policy that provides the largest capacity for both scale in and scale out.
2. **Customized Metrics for Target Tracking** – Be cautious when using customized metrics for Target Tracking as Auto scaling is best suited to scale-out/in proportional to changes in metrics chosen for the policy. If those metrics don't change proportionally to the scaling actions used for policy creation, it might lead to continuous scale-out or scale-in actions which might affect availability or cost.

   For data tiering clusters (r6gd, r7gd, and r8gd), avoid using memory-based metrics for scaling.
3. **Scheduled Scaling** – If you identify that your workload is deterministic (reaches high/low at a specific time), we recommend using Scheduled Scaling and configure your target capacity according to the need. Target Tracking is best suitable for non-deterministic workloads and for the cluster to operate at the required target metric by scaling out when you need more resources and scaling in when you need less.
4. **Disable Scale-In** – Auto scaling on Target Tracking is best suited for clusters with gradual increase/decrease of workloads as spikes/dip in metrics can trigger consecutive scale-out/in oscillations. In order to avoid such oscillations, you can start with scale-in disabled and later you can always manually scale-in to your need.
5. **Test your application** – We recommend you test your application with your estimated Min/Max workloads to determine absolute Min,Max shards/replicas required for the cluster while creating Scaling policies to avoid availability issues. Auto scaling can scale out to the Max and scale-in to the Min threshold configured for the target.

6. **Defining Target Value** – You can analyze corresponding CloudWatch metrics for cluster utilization over a four-week period to determine the target value threshold. If you are still not sure of of what value to choose, we recommend starting with a minimum supported predefined metric value.
7. AutoScaling on Target Tracking is best suited for clusters with uniform distribution of workloads across shards/replicas dimension. Having non-uniform distribution can lead to:
   - Scaling when not required due to workload spike/dip on a few hot shards/replicas.
   - Not scaling when required due to overall avg close to target even though having hot shards/replicas.

> **Note**
>
> When scaling out your cluster, ElastiCache will automatically replicate the Functions loaded in one of the existing nodes (selected at random) to the new node(s). If your cluster has Valkey or Redis OSS 7.0 or above and your application uses Functions, we recommend loading all of your functions to all the shards before scaling out so that your cluster does not end up with different functions on different shards.

After registering to AutoScaling, note the following:

- There are limitations on Auto scaling Supported Configurations, so we recommend you not change configuration of a replication group that is registered for Auto scaling. The following are examples:
  - Manually modifying instance type to unsupported types.
  - Associating the replication group to a Global datastore.
  - Changing `ReservedMemoryPercent` parameter.
  - Manually increasing/decreasing shards/replicas beyond the Min/Max capacity configured during policy creation.

## Using Auto Scaling with shards

With ElastiCache's AutoScaling you can use tracking and scheduled policies with your Valkey or Redis OSS engine.

The following provides details on target tracking and scheduled policies and how to apply them using the AWS Management Console AWS CLI and APIs.

**Topics**


- Target tracking scaling policies
- Adding a scaling policy
- Registering a Scalable Target
- Defining a scaling policy
- Disabling scale-in activity
- Applying a scaling policy
- Editing a scaling policy
- Deleting a scaling policy
- Use CloudFormation for Auto Scaling policies
- Scheduled scaling


### Target tracking scaling policies

With target tracking scaling policies, you select a metric and set a target value. ElastiCache for Valkey and Redis OSS Auto Scaling creates and manages the CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and the target value. The scaling policy adds or removes shards as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target tracking scaling policy also adjusts to the fluctuations in the metric due to a fluctuating load pattern and minimizes rapid fluctuations in the capacity of the fleet.

For example, consider a scaling policy that uses the predefined average `ElastiCachePrimaryEngineCPUUtilization` metric with configured target value. Such a policy can keep CPU utilization at, or close to the specified target value.

### Predefined metrics

A predefined metric is a structure that refers to a specific name, dimension, and statistic (`average`) of a given CloudWatch metric. Your Auto Scaling policy defines one of the below predefined metrics for your cluster:

| Predefined Metric Name | CloudWatch Metric Name | CloudWatch Metric Dimension | Ineligible Instance Types |
|---|---|---|---|
| ElastiCachePrimaryEngineCPUUtilization | EngineCPUUtilization | ReplicationGroupId, Role = Primary | None |
| ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage | DatabaseCapacityUsageCountedForEvictPercentage | Valkey or Redis OSS Replication Group Metrics | None |
| ElastiCacheDatabaseMemoryUsageCountedForEvictPercentage | DatabaseMemoryUsageCountedForEvictPercentage | Valkey or Redis OSS Replication Group Metrics | R6gd, R7gd, R8gd |

Data-tiered instance types cannot use `ElastiCacheDatabaseMemoryUsageCountedForEvictPercentage`, as these instance types store data in both memory and SSD. The expected use case for data-tiered instances is to have 100 percent memory usage and fill up SSD as needed.

**Auto Scaling criteria for shards**

When the service detects that your predefined metric is equal to or greater than the Target setting, it will increase your shards capacity automatically. ElastiCache for Valkey and Redis OSS scales out your cluster shards by a count equal to the larger of two numbers: Percent variation from Target

and 20 percent of current shards. For scale-in, ElastiCache won't auto scale-in unless the overall metric value is below 75 percent of your defined Target.

For a scale out example, if you have 50 shards and

- if your Target breaches by 30 percent, ElastiCache scales out by 30 percent, which results in 65 shards per cluster.
- if your Target breaches by 10 percent, ElastiCache scales out by default Minimum of 20 percent, which results in 60 shards per cluster.

For a scale-in example, if you have selected a Target value of 60 percent, ElastiCache won't auto scale-in until the metric is less than or equal to 45 percent (25 percent below the Target 60 percent).

**Auto Scaling considerations**

Keep the following considerations in mind:

- A target tracking scaling policy assumes that it should perform scale out when the specified metric is above the target value. You cannot use a target tracking scaling policy to scale out when the specified metric is below the target value. ElastiCache for Valkey and Redis OSS scales out shards by a minimum of 20 percent deviation of target of existing shards in the cluster.
- A target tracking scaling policy does not perform scaling when the specified metric has insufficient data. It does not perform scale-in because it does not interpret insufficient data as low utilization.
- You may see gaps between the target value and the actual metric data points. This is because ElastiCache Auto Scaling always acts conservatively by rounding up or down when it determines how much capacity to add or remove. This prevents it from adding insufficient capacity or removing too much capacity.
- To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more conservatively.
- You can have multiple target tracking scaling policies for an ElastiCache for Valkey and Redis OSS cluster, provided that each of them uses a different metric. The intention of ElastiCache Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the service if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion enabled) are ready to scale in.

- Do not edit or delete the CloudWatch alarms that ElastiCache Auto Scaling manages for a target tracking scaling policy. ElastiCache Auto Scaling deletes the alarms automatically when you delete the scaling policy.
- ElastiCache Auto Scaling doesn't prevent you from manually modifying cluster shards. These manual adjustments don't affect any existing CloudWatch alarms that are attached to the scaling policy but can impact metrics that may trigger these CloudWatch alarms.
- These CloudWatch alarms managed by Auto Scaling are defined over the AVG metric across all the shards in the cluster. So, having hot shards can result in either scenario of:
  - scaling when not required due to load on a few hot shards triggering a CloudWatch alarm
  - not scaling when required due to aggregated AVG across all shards affecting alarm not to breach.
- ElastiCache default limits on Nodes per cluster still applies. So, when opting for Auto Scaling and if you expect maximum nodes to be more than default limit, request a limit increase at AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.
- Ensure that you have enough ENIs (Elastic Network Interfaces) available in your VPC, which are required during scale-out. For more information, see Elastic network interfaces.
- If there is not enough capacity available from EC2, ElastiCache Auto Scaling would not scale and be delayed til the capacity is available.
- ElastiCache for Redis OSS Auto Scaling during scale-in will not remove shards with slots having an item size larger than 256 MB post-serialization.
- During scale-in it will not remove shards if insufficient memory available on resultant shard configuration.

### Adding a scaling policy

You can add a scaling policy using the AWS Management Console.

#### To add an Auto Scaling policy to an ElastiCache for Valkey and Redis OSS cluster

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**.
3. Choose the cluster that you want to add a policy to (choose the cluster name and not the button to its left).
4. Choose the **Auto Scaling policies** tab.

5. Choose **add dynamic scaling**.
6. For **Policy name** enter a policy name.
7. For **Scalable Dimension** choose **shards**.
8. For the target metric, choose one of the following:

   - **Primary CPU Utilization** to create a policy based on the average CPU utilization.
   - **Memory** to create a policy based on the average database memory.
   - **Capacity** to create a policy based on average database capacity usage. The Capacity metric includes memory and SSD utilization for data-tiered instances, and memory utilization for all other instance types.
9. For the target value, choose a value greater than or equal to 35 and less than or equal to 70. Auto scaling will maintain this value for the selected target metric across your ElastiCache shards:

   - **Primary CPU Utilization**: maintains target value for `EngineCPUUtilization` metric on primary nodes.
   - **Memory**: maintains target value for `DatabaseMemoryUsageCountedForEvictPercentage` metric
   - **Capacity** maintains target value for `DatabaseCapacityUsageCountedForEvictPercentage` metric,

   Cluster shards are added or removed to keep the metric close to the specified value.
10. (Optional) Scale-in or scale-out cooldown periods are not supported from the console. Use the AWS CLI to modify the cooldown values.
11. For **Minimum capacity**, type the minimum number of shards that the ElastiCache Auto Scaling policy is required to maintain.
12. For **Maximum capacity**, type the maximum number of shards that the ElastiCache Auto Scaling policy is required to maintain. This value must be less than or equal to 250.
13. Choose **Create**.

### Registering a Scalable Target

Before you can use Auto Scaling with an ElastiCache for Valkey and Redis OSS cluster, you register your cluster with ElastiCache auto scaling. You do so to define the scaling dimension and limits

to be applied to that cluster. ElastiCache auto scaling dynamically scales the cluster along the `elasticache:replication-group:NodeGroups` scalable dimension, which represents the number of cluster shards.

### Using the AWS CLI

To register your ElastiCache for Valkey and Redis OSS cluster, use the register-scalable-target command with the following parameters:

- `--service-namespace` – Set this value to `elasticache`
- `--resource-id` – The resource identifier for the cluster. For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- `--scalable-dimension` – Set this value to `elasticache:replication-group:NodeGroups`.
- `--max-capacity` – The maximum number of shards to be managed by ElastiCache auto scaling. For information about the relationship between `--min-capacity`, `--max-capacity`, and the number of shards in your cluster, see Minimum and maximum capacity.
- `--min-capacity` – The minimum number of shards to be managed by ElastiCache auto scaling. For information about the relationship between `--min-capacity`, `--max-capacity`, and the number of shards in your cluster, see Minimum and maximum capacity.

### Example

In the following example, you register an ElastiCache cluster named `myscalablecluster`. The registration indicates that the cluster should be dynamically scaled to have from one to ten shards.

For Linux, macOS, or Unix:

```
aws application-autoscaling register-scalable-target \
    --service-namespace elasticache \
    --resource-id replication-group/myscalablecluster \
    --scalable-dimension elasticache:replication-group:NodeGroups \
    --min-capacity 1 \
    --max-capacity 10 \
```

For Windows:

```
aws application-autoscaling register-scalable-target ^
```

```
    --service-namespace elasticache ^
    --resource-id replication-group/myscalablecluster ^
    --scalable-dimension elasticache:replication-group:NodeGroups ^
    --min-capacity 1 ^
    --max-capacity 10 ^
```

### Using the API

To register your ElastiCache cluster, use the register-scalable-target command with the following parameters:

- ServiceNamespace – Set this value to elasticache.
- ResourceID – The resource identifier for the ElastiCache cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- ScalableDimension – Set this value to `elasticache:replication-group:NodeGroups`.
- MinCapacity – The minimum number of shards to be managed by ElastiCache auto scaling. For information about the relationship between --min-capacity, --max-capacity, and the number of replicas in your cluster, see Minimum and maximum capacity.
- MaxCapacity – The maximum number of shards to be managed by ElastiCache auto scaling. For information about the relationship between --min-capacity, --max-capacity, and the number of replicas in your cluster, see Minimum and maximum capacity.

### Example

In the following example, you register an ElastiCache cluster named `myscalablecluster` with the Application Auto Scaling API. This registration indicates that the cluster should be dynamically scaled to have from one to 5 replicas.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS
{
```

```
    "ServiceNamespace": "elasticache",
    "ResourceId": "replication-group/myscalablecluster",
    "ScalableDimension": "elasticache:replication-group:NodeGroups",
    "MinCapacity": 1,
    "MaxCapacity": 5
}
```

### Defining a scaling policy

A target-tracking scaling policy configuration is represented by a JSON block that the metrics and target values are defined in. You can save a scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see TargetTrackingScalingPolicyConfiguration in the Application Auto Scaling API Reference.

The following options are available for defining a target-tracking scaling policy configuration:

**Topics**

- Using a predefined metric
- Using a custom metric
- Using cooldown periods

### Using a predefined metric

By using predefined metrics, you can quickly define a target-tracking scaling policy for an ElastiCache for Valkey and Redis OSS cluster that works with target tracking in ElastiCache Auto Scaling.

Currently, ElastiCache supports the following predefined metrics in NodeGroup Auto Scaling:

- **ElastiCachePrimaryEngineCPUUtilization** – The average value of the `EngineCPUUtilization` metric in CloudWatch across all primary nodes in the cluster.
- **ElastiCacheDatabaseMemoryUsageCountedForEvictPercentage** – The average value of the `DatabaseMemoryUsageCountedForEvictPercentage` metric in CloudWatch across all primary nodes in the cluster.
- **ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage** – The average value of the `ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage` metric in CloudWatch across all primary nodes in the cluster.

For more information about the `EngineCPUUtilization`, `DatabaseMemoryUsageCountedForEvictPercentage` and `DatabaseCapacityUsageCountedForEvictPercentage` metrics, see Monitoring use with CloudWatch Metrics. To use a predefined metric in your scaling policy, you create a target tracking configuration for your scaling policy. This configuration must include a `PredefinedMetricSpecification` for the predefined metric and a TargetValue for the target value of that metric.

**Example**

The following example describes a typical policy configuration for target-tracking scaling for an ElastiCache for Valkey and Redis OSS cluster. In this configuration, the `ElastiCachePrimaryEngineCPUUtilization` predefined metric is used to adjust the cluster based on an average CPU utilization of 40 percent across all primary nodes in the cluster.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "ElastiCachePrimaryEngineCPUUtilization"
    }
}
```

**Using a custom metric**

By using custom metrics, you can define a target-tracking scaling policy that meets your custom requirements. You can define a custom metric based on any ElastiCache metric that changes in proportion to scaling. Not all ElastiCache metrics work for target tracking. The metric must be a valid utilization metric and describe how busy an instance is. The value of the metric must increase or decrease in proportion to the number of Shards in the cluster. This proportional increase or decrease is necessary to use the metric data to proportionally scale out or in the number of shards.

**Example**

The following example describes a target-tracking configuration for a scaling policy. In this configuration, a custom metric adjusts an ElastiCache for Redis OSS cluster based on an average CPU utilization of 50 percent across all shards in an cluster named `my-db-cluster`.

```
{
```

```
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "EngineCPUUtilization",
        "Namespace": "AWS/ElastiCache",
        "Dimensions": [
            {
                "Name": "ReplicationGroup","Value": "my-db-cluster"
            },
            {
                "Name": "Role","Value": "PRIMARY"
            }
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

### Using cooldown periods

You can specify a value, in seconds, for `ScaleOutCooldown` to add a cooldown period for scaling out your cluster. Similarly, you can add a value, in seconds, for `ScaleInCooldown` to add a cooldown period for scaling in your cluster. For more information, see TargetTrackingScalingPolicyConfiguration in the Application Auto Scaling API Reference.

The following example describes a target-tracking configuration for a scaling policy. In this configuration, the `ElastiCachePrimaryEngineCPUUtilization` predefined metric is used to adjust an ElastiCache for Redis OSS cluster based on an average CPU utilization of 40 percent across all primary nodes in that cluster. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "ElastiCachePrimaryEngineCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

### Disabling scale-in activity

You can prevent the target-tracking scaling policy configuration from scaling in your cluster by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting shards, while still allowing the scaling policy to create them as needed.

You can specify a Boolean value for `DisableScaleIn` to enable or disable scale in activity for your cluster. For more information, see TargetTrackingScalingPolicyConfiguration in the Application Auto Scaling API Reference.

The following example describes a target-tracking configuration for a scaling policy. In this configuration, the `ElastiCachePrimaryEngineCPUUtilization` predefined metric adjusts an ElastiCache for Valkey and Redis OSS cluster based on an average CPU utilization of 40 percent across all primary nodes in that cluster. The configuration disables scale-in activity for the scaling policy.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "ElastiCachePrimaryEngineCPUUtilization"
    },
    "DisableScaleIn": true
}
```

### Applying a scaling policy

After registering your cluster with ElastiCache for Valkey and Redis OSS auto scaling and defining a scaling policy, you apply the scaling policy to the registered cluster. To apply a scaling policy to an ElastiCache for Redis OSS cluster, you can use the AWS CLI or the Application Auto Scaling API.

### Applying a scaling policy using the AWS CLI

To apply a scaling policy to your ElastiCache for Valkey and Redis OSS cluster, use the put-scaling-policy command with the following parameters:

- **--policy-name** – The name of the scaling policy.
- **--policy-type** – Set this value to `TargetTrackingScaling`.
- **--resource-id** – The resource identifier. For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.

- **--service-namespace** – Set this value to `elasticache`.
- **--scalable-dimension** – Set this value to `elasticache:replication-group:NodeGroups`.
- **--target-tracking-scaling-policy-configuration** – The target-tracking scaling policy configuration to use for the cluster.

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an ElastiCache for Valkey and Redis OSS cluster named `myscalablecluster` with ElastiCache auto scaling. To do so, you use a policy configuration saved in a file named `config.json`.

For Linux, macOS, or Unix:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id replication-group/myscalablecluster \
    --service-namespace elasticache \
    --scalable-dimension elasticache:replication-group:NodeGroups \
    --target-tracking-scaling-policy-configuration file://config.json
```

For Windows:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id replication-group/myscalablecluster ^
    --service-namespace elasticache ^
    --scalable-dimension elasticache:replication-group:NodeGroups ^
    --target-tracking-scaling-policy-configuration file://config.json
```

**Applying a scaling policy using the API**

To apply a scaling policy to your ElastiCache for Valkey and Redis OSS cluster, use the PutScalingPolicy AWS CLI command with the following parameters:

- **--policy-name** – The name of the scaling policy.
- **--resource-id** – The resource identifier. For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.

- **--service-namespace** – Set this value to `elasticache`.
- **--scalable-dimension** – Set this value to `elasticache:replication-group:NodeGroups`.
- **--target-tracking-scaling-policy-configuration** – The target-tracking scaling policy configuration to use for the cluster.

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an ElastiCache cluster named `myscalablecluster` with ElastiCache auto scaling. You use a policy configuration based on the `ElastiCachePrimaryEngineCPUUtilization` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS
{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "elasticache",
    "ResourceId": "replication-group/myscalablecluster",
    "ScalableDimension": "elasticache:replication-group:NodeGroups",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "ElastiCachePrimaryEngineCPUUtilization"
        }
    }
}
```

### Editing a scaling policy

You can edit a scaling policy using the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

### Editing a scaling policy using the AWS Management Console

#### To edit an Auto Scaling policy for an ElastiCache for Valkey and Redis OSS cluster

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose the appropriate engine.
3. Choose the cluster that you want to add a policy to (choose the cluster name and not the button to its left).
4. Choose the **Auto Scaling policies** tab.
5. Under **Scaling policies**, choose the button to the left of the Auto Scaling policy you wish to change, and then choose **Modify**.
6. Make the requisite changes to the policy.
7. Choose **Modify**.

### Editing a scaling policy using the AWS CLI and API

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- When using the AWS CLI, specify the name of the policy you want to edit in the `--policy-name` parameter. Specify new values for the parameters you want to change.
- When using the Application Auto Scaling API, specify the name of the policy you want to edit in the `PolicyName` parameter. Specify new values for the parameters you want to change.

For more information, see Applying a scaling policy.

### Deleting a scaling policy

You can delete a scaling policy using the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

### Deleting a scaling policy using the AWS Management Console

#### To delete an Auto Scaling policy for an ElastiCache for Redis OSS cluster

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. In the navigation pane, choose **Valkey** or **Redis OSS**.
3. Choose the cluster whose Auto Scaling policy you want to edit (choose the cluster name, not the button to its left).
4. Choose the **Auto Scaling policies** tab.
5. Under **Scaling policies**, choose the Auto Scaling policy, and then choose **Delete**.

### Deleting a scaling policy using the AWS CLI

To delete a scaling policy to your ElastiCache for Valkey and Redis OSS cluster, use the delete-scaling-policy AWS CLI command with the following parameters:

- **--policy-name** – The name of the scaling policy.
- **--resource-id** – The resource identifier. For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- **--service-namespace** – Set this value to `elasticache`.
- **--scalable-dimension** – Set this value to `elasticache:replication-group:NodeGroups`.

In the following example, you delete a target-tracking scaling policy named `myscalablepolicy` from a cluster named `myscalablecluster`.

For Linux, macOS, or Unix:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id replication-group/myscalablecluster \
    --service-namespace elasticache \
    --scalable-dimension elasticache:replication-group:NodeGroups
```

For Windows:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id replication-group/myscalablecluster ^
    --service-namespace elasticache ^
    --scalable-dimension elasticache:replication-group:NodeGroups
```

### Deleting a scaling policy using the API

To delete a scaling policy to your ElastiCache for Valkey and Redis OSS cluster, use the DeleteScalingPolicy AWS CLI command with the following parameters:

- **--policy-name** – The name of the scaling policy.
- **--resource-id** – The resource identifier. For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- **--service-namespace** – Set this value to `elasticache`.
- **--scalable-dimension** – Set this value to `elasticache:replication-group:NodeGroups`.

In the following example, you delete a target-tracking scaling policy named `myscalablepolicy` from a cluster named `myscalablecluster`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS
{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "elasticache",
    "ResourceId": "replication-group/myscalablecluster",
    "ScalableDimension": "elasticache:replication-group:NodeGroups"
}
```

### Use CloudFormation for Auto Scaling policies

This snippet shows how to create a target tracking policy and apply it to an AWS::ElastiCache::ReplicationGroup resource using the AWS::ApplicationAutoScaling::ScalableTarget resource. It uses the Fn::Join and Ref intrinsic functions to construct the `ResourceId` property with the logical name of the `AWS::ElastiCache::ReplicationGroup` resource that is specified in the same template.

```
ScalingTarget:
```

```
    Type: 'AWS::ApplicationAutoScaling::ScalableTarget'
    Properties:
      MaxCapacity: 3
      MinCapacity: 1
      ResourceId: !Sub replication-group/${logicalName}
      ScalableDimension: 'elasticache:replication-group:NodeGroups'
      ServiceNamespace: elasticache
      RoleARN: !Sub "arn:aws:iam::${AWS::AccountId}:role/aws-
service-role/elasticache.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG"

  ScalingPolicy:
    Type: "AWS::ApplicationAutoScaling::ScalingPolicy"
    Properties:
      ScalingTargetId: !Ref ScalingTarget
      ServiceNamespace: elasticache
      PolicyName: testpolicy
      PolicyType: TargetTrackingScaling
      ScalableDimension: 'elasticache:replication-group:NodeGroups'
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ElastiCachePrimaryEngineCPUUtilization
        TargetValue: 40
```

### Scheduled scaling

Scaling based on a schedule enables you to scale your application in response to predictable changes in demand. To use scheduled scaling, you create scheduled actions, which tell ElastiCache for Valkey and Redis OSS to perform scaling activities at specific times. When you create a scheduled action, you specify an existing cluster, when the scaling activity should occur, minimum capacity, and maximum capacity. You can create scheduled actions that scale one time only or that scale on a recurring schedule.

You can only create a scheduled action for clusters that already exist. You can't create a scheduled action at the same time that you create a cluster.

For more information on terminology for scheduled action creation, management, and deletion, see Commonly used commands for scheduled action creation, management, and deletion

**To create on a recurring schedule:**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**.
3. Choose the cluster that you want to add a policy for.
4. Choose the **Manage Auto Scaling policie** from the **Actions** dropdown.
5. Choose the **Auto Scaling policies** tab.
6. In the **Auto scaling policies** section, the **Add Scaling policy** dialog box appears. Choose **Scheduled scaling**.
7. For **Policy Name**, enter the policy name.
8. For **Scalable Dimension**, choose **Shards**.
9. For **Target Shards**, choose the value.
10. For **Recurrence**, choose **Recurring**.
11. For **Frequency**, choose the respective value.
12. For **Start Date** and **Start time**, choose the time from when the policy will go into effect.
13. Choose **Add Policy**.

**To create a one-time scheduled action:**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**.
3. Choose the cluster that you want to add a policy for.
4. Choose the **Manage Auto Scaling policie** from the **Actions** dropdown.
5. Choose the **Auto Scaling policies** tab.
6. In the **Auto scaling policies** section, the **Add Scaling policy** dialog box appears. Choose **Scheduled scaling**.
7. For **Policy Name**, enter the policy name.
8. For **Scalable Dimension**, choose **Shards**.
9. For **Target Shards**, choose the value.

10. For **Recurrence**, choose **One Time**.
11. For **Start Date** and **Start time**, choose the time from when the policy will go into effect.
12. For **End Date** choose the date until when the policy would be in effect.
13. Choose **Add Policy**.

#### To delete a scheduled action

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**.
3. Choose the cluster that you want to add a policy for.
4. Choose the **Manage Auto Scaling policie** from the **Actions** dropdown.
5. Choose the **Auto Scaling policies** tab.
6. In the **Auto scaling policies** section, choose the auto scaling policy, and then choose **Delete** from the **Actions** dialog.

#### To manage scheduled scaling using the AWS CLI

Use the following application-autoscaling APIs:

- put-scheduled-action
- describe-scheduled-actions
- delete-scheduled-action

#### Use CloudFormation to create a scheduled action

This snippet shows how to create a target tracking policy and apply it to an AWS::ElastiCache::ReplicationGroup resource using the AWS::ApplicationAutoScaling::ScalableTarget resource. It uses the Fn::Join and Ref intrinsic functions to construct the `ResourceId` property with the logical name of the `AWS::ElastiCache::ReplicationGroup` resource that is specified in the same template.

```
ScalingTarget:
   Type: 'AWS::ApplicationAutoScaling::ScalableTarget'
```

```
   Properties:
     MaxCapacity: 3
     MinCapacity: 1
     ResourceId: !Sub replication-group/${logicalName}
     ScalableDimension: 'elasticache:replication-group:NodeGroups'
     ServiceNamespace: elasticache
     RoleARN: !Sub "arn:aws:iam::${AWS::AccountId}:role/aws-
service-role/elasticache.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG"
     ScheduledActions:
       - EndTime: '2020-12-31T12:00:00.000Z'
         ScalableTargetAction:
           MaxCapacity: '5'
           MinCapacity: '2'
         ScheduledActionName: First
         Schedule: 'cron(0 18 * * ? *)'
```

## Using Auto Scaling with replicas

An ElastiCache replication group can set up one or more caches to work as a single logical node.

The following provides details on target tracking and scheduled policies and how to apply them using the AWS Management Console AWS CLI and APIs.

### Target tracking scaling policies

With target tracking scaling policies, you select a metric and set a target value. ElastiCache for Valkey and Redis OSS AutoScaling creates and manages the CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and the target value. The scaling policy adds or removes replicas uniformly across all shards as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target tracking scaling policy also adjusts to the fluctuations in the metric due to a fluctuating load pattern and minimizes rapid fluctuations in the capacity of the fleet.

### Auto Scaling criteria for replicas

Your Auto Scaling policy defines the following predefined metric for your cluster:

`ElastiCacheReplicaEngineCPUUtilization`: The AVG EngineCPU utilization threshold aggregated across all replicas that ElastiCache uses to trigger an auto-scaling operation. You can set the utilization target between 35 percent and 70 percent.

When the service detects that your `ElastiCacheReplicaEngineCPUUtilization` metric is equal to or greater than the Target setting, it will increase replicas across your shards automatically. ElastiCache scales out your cluster replicas by a count equal to the larger of two numbers: Percent variation from Target and one replica. For scale-in, ElastiCache won't auto scale-in unless the overall metric value is below 75 percent of your defined Target.

For a scale-out example, if you have 5 shards and 1 replica each:

If your Target breaches by 30 percent, ElastiCache for Valkey and Redis OSS scales out by 1 replica (max(0.3, default 1)) across all shards. which results in 5 shards with 2 replicas each,

For a scale-in example, if you have selected Target value of 60 percent, ElastiCache for Valkey and Redis OSS won't auto scale-in until the metric is less than or equal to 45 percent (25 percent below the Target 60 percent).

**Auto Scaling considerations**

Keep the following considerations in mind:

- A target tracking scaling policy assumes that it should perform scale out when the specified metric is above the target value. You cannot use a target tracking scaling policy to scale out when the specified metric is below the target value. ElastiCache for Valkey and Redis OSS scales out replicas by maximum of (% deviation rounded off from Target, default 1) of existing replicas across all shards in the cluster.
- A target tracking scaling policy does not perform scaling when the specified metric has insufficient data. It does not perform scale in because it does not interpret insufficient data as low utilization.
- You may see gaps between the target value and the actual metric data points. This is because ElastiCache Auto Scaling always acts conservatively by rounding up or down when it determines how much capacity to add or remove. This prevents it from adding insufficient capacity or removing too much capacity.
- To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more gradually with max scale in of 1 replica across the shards in the cluster.
- You can have multiple target tracking scaling policies for an ElastiCache for Valkey and Redis OSS cluster, provided that each of them uses a different metric. The intention of Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the service if any of the target tracking

policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion enabled) are ready to scale in.

- Do not edit or delete the CloudWatch alarms that ElastiCache Auto Scaling manages for a target tracking scaling policy. Auto Scaling deletes the alarms automatically when you delete the scaling policy or deleting the cluster.
- ElastiCache Auto Scaling doesn't prevent you from manually modifying replicas across shards. These manual adjustments don't affect any existing CloudWatch alarms that are attached to the scaling policy but can impact metrics that may trigger these CloudWatch alarms.
- These CloudWatch alarms managed by Auto Scaling are defined over the AVG metric across all the shards in the cluster. So, having hot shards can result in either scenario of:
  - scaling when not required due to load on a few hot shards triggering a CloudWatch alarm
  - not scaling when required due to aggregated AVG across all shards affecting alarm not to breach.
- ElastiCache default limits on nodes per cluster still applies. So, when opting for Auto Scaling and if you expect maximum nodes to be more than default limit, request a limit increase at AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.
- Ensure that you have enough ENIs (Elastic Network Interfaces) available in your VPC, which are required during scale-out. For more information, see Elastic network interfaces.
- If there is not enough capacity available from EC2, ElastiCache Auto Scaling will not scale out until either the capacity is available, or if you manually modify the cluster to the instance types that have enough capacity.
- ElastiCache Auto Scaling doesn't support scaling of replicas with a cluster having `ReservedMemoryPercent` less than 25 percent. For more information, see Managing reserved memory for Valkey and Redis OSS.

### Adding a scaling policy

You can add a scaling policy using the AWS Management Console.

### Adding a scaling policy using the AWS Management Console

To add an auto scaling policy to ElastiCache for Valkey and Redis OSS

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**.

3. Choose the cluster that you want to add a policy to (choose the cluster name and not the button to its left).
4. Choose the **Auto Scaling policies** tab.
5. Choose **add dynamic scaling**.
6. Under **Scaling policies**, choose **Add dynamic scaling**.
7. For **Policy Name**, enter the policy name.
8. For **Scalable Dimension**, select **Replicas** from dialog box.
9. For the target value, type the Avg percentage of CPU utilization that you want to maintain on ElastiCache Replicas. This value must be >=35 and <=70. Cluster replicas are added or removed to keep the metric close to the specified value.
10. (Optional) scale-in or scale-out cooldown periods are not supported from the Console. Use the AWS CLI to modify the cool down values.
11. For **Minimum capacity**, type the minimum number of replicas that the ElastiCache Auto Scaling policy is required to maintain.
12. For **Maximum capacity**, type the maximum number of replicas the ElastiCache Auto Scaling policy is required to maintain. This value must be >=5.
13. Choose **Create**.

### Registering a Scalable Target

You can apply a scaling policy based on either a predefined or custom metric. To do so, you can use the AWS CLI or the Application Auto Scaling API. The first step is to register your ElastiCache for Valkey and Redis OSS replication group with Auto Scaling.

Before you can use ElastiCache auto scaling with a cluster, you must register your cluster with ElastiCache auto scaling. You do so to define the scaling dimension and limits to be applied to that cluster. ElastiCache auto scaling dynamically scales the cluster along the `elasticache:replication-group:Replicas` scalable dimension, which represents the number of cluster replicas per shard.

### Using the CLI

To register your ElastiCache cluster, use the register-scalable-target command with the following parameters:

- --service-namespace – Set this value to elasticache.

- --resource-id – The resource identifier for the ElastiCache cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- --scalable-dimension – Set this value to `elasticache:replication-group:Replicas`.
- --min-capacity – The minimum number of replicas to be managed by ElastiCache auto scaling. For information about the relationship between --min-capacity, --max-capacity, and the number of replicas in your cluster, see Minimum and maximum capacity.
- --max-capacity – The maximum number of replicas to be managed by ElastiCache auto scaling. For information about the relationship between --min-capacity, --max-capacity, and the number of replicas in your cluster, see Minimum and maximum capacity.

#### Example

In the following example, you register an ElastiCache cluster named `myscalablecluster`. The registration indicates that the cluster should be dynamically scaled to have from one to 5 replicas.

For Linux, macOS, or Unix:

```
aws application-autoscaling register-scalable-target \
    --service-namespace elasticache \
    --resource-id replication-group/myscalablecluster \
    --scalable-dimension elasticache:replication-group:Replicas \
    --min-capacity 1 \
    --max-capacity 5 \
```

For Windows:

```
aws application-autoscaling register-scalable-target ^
    --service-namespace elasticache ^
    --resource-id replication-group/myscalablecluster ^
    --scalable-dimension elasticache:replication-group:Replicas ^
    --min-capacity 1 ^
    --max-capacity 5 ^
```

#### Using the API

To register your ElastiCache cluster, use the register-scalable-target command with the following parameters:

- ServiceNamespace – Set this value to elasticache.
- ResourceID – The resource identifier for the ElastiCache cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- ScalableDimension – Set this value to `elasticache:replication-group:Replicas`.
- MinCapacity – The minimum number of replicas to be managed by ElastiCache auto scaling. For information about the relationship between --min-capacity, --max-capacity, and the number of replicas in your cluster, see Minimum and maximum capacity.
- MaxCapacity – The maximum number of replicas to be managed by ElastiCache auto scaling. For information about the relationship between --min-capacity, --max-capacity, and the number of replicas in your cluster, see Minimum and maximum capacity.

### Example

In the following example, you register a cluster named `myscalablecluster` with the Application Auto Scaling API. This registration indicates that the cluster should be dynamically scaled to have from one to 5 replicas.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS
{
    "ServiceNamespace": "elasticache",
    "ResourceId": "replication-group/myscalablecluster",
    "ScalableDimension": "elasticache:replication-group:Replicas",
    "MinCapacity": 1,
    "MaxCapacity": 5
}
```

### Defining a scaling policy

A target-tracking scaling policy configuration is represented by a JSON block that the metrics and target values are defined in. You can save a scaling policy configuration as a JSON block in a text

file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see TargetTrackingScalingPolicyConfiguration in the *Application Auto Scaling API Reference*.

The following options are available for defining a target-tracking scaling policy configuration:

**Topics**


- Using a predefined metric
- Editing a scaling policy
- Deleting a scaling policy
- Use CloudFormation for Auto Scaling policies
- Scheduled scaling


### Using a predefined metric

A target-tracking scaling policy configuration is represented by a JSON block that the metrics and target values are defined in. You can save a scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see TargetTrackingScalingPolicyConfiguration in the *Application Auto Scaling API Reference*.

The following options are available for defining a target-tracking scaling policy configuration:

**Topics**


- Using a predefined metric
- Using a custom metric
- Using cooldown periods
- Disabling scale-in activity
- Applying a scaling policy to an ElastiCache for Valkey and Redis OSS cluster


### Using a predefined metric

By using predefined metrics, you can quickly define a target-tracking scaling policy for an ElastiCache for Valkey and Redis OSS cluster that works with target tracking in ElastiCache Auto Scaling. Currently, ElastiCache supports the following predefined metric in ElastiCache Replicas Auto Scaling:

`ElastiCacheReplicaEngineCPUUtilization` – The average value of the EngineCPUUtilization metric in CloudWatch across all replicas in the cluster. You can find the aggregated metric value in CloudWatch under ElastiCache `ReplicationGroupId, Role` for required ReplicationGroupId and Role Replica.

To use a predefined metric in your scaling policy, you create a target tracking configuration for your scaling policy. This configuration must include a `PredefinedMetricSpecification` for the predefined metric and a `TargetValue` for the target value of that metric.

**Using a custom metric**

By using custom metrics, you can define a target-tracking scaling policy that meets your custom requirements. You can define a custom metric based on any ElastiCache for Valkey and Redis OSS metric that changes in proportion to scaling. Not all ElastiCache metrics work for target tracking. The metric must be a valid utilization metric and describe how busy an instance is. The value of the metric must increase or decrease in proportion to the number of replicas in the cluster. This proportional increase or decrease is necessary to use the metric data to proportionally increase or decrease the number of replicas.

**Example**

The following example describes a target-tracking configuration for a scaling policy. In this configuration, a custom metric adjusts a cluster based on an average CPU utilization of 50 percent across all replicas in an cluster named `my-db-cluster`.

```
{"TargetValue": 50,
    "CustomizedMetricSpecification":
    {"MetricName": "EngineCPUUtilization",
        "Namespace": "AWS/ElastiCache",
        "Dimensions": [
            {"Name": "ReplicationGroup","Value": "my-db-cluster"},
            {"Name": "Role","Value": "REPLICA"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

### Using cooldown periods

You can specify a value, in seconds, for `ScaleOutCooldown` to add a cooldown period for scaling out your cluster. Similarly, you can add a value, in seconds, for `ScaleInCooldown` to add a cooldown period for scaling in your cluster. For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see TargetTrackingScalingPolicyConfiguration in the *Application Auto Scaling API Reference*. The following example describes a target-tracking configuration for a scaling policy. In this configuration, the `ElastiCacheReplicaEngineCPUUtilization`predefined metric is used to adjust a cluster based on an average CPU utilization of 40 percent across all replicas in that cluster. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```
{"TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {"PredefinedMetricType": "ElastiCacheReplicaEngineCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

### Disabling scale-in activity

You can prevent the target-tracking scaling policy configuration from scaling in your ElastiCache for Valkey and Redis OSS cluster by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting replicas, while still allowing the scaling policy to add them as needed.

You can specify a Boolean value for `DisableScaleIn` to enable or disable scale in activity for your cluster. For more information about `DisableScaleIn`, see TargetTrackingScalingPolicyConfiguration in the *Application Auto Scaling API Reference*.

### Example

The following example describes a target-tracking configuration for a scaling policy. In this configuration, the `ElastiCacheReplicaEngineCPUUtilization` predefined metric adjusts a cluster based on an average CPU utilization of 40 percent across all replicas in that cluster. The configuration disables scale-in activity for the scaling policy.

```
{"TargetValue": 40.0,
```

```
    "PredefinedMetricSpecification":
    {"PredefinedMetricType": "ElastiCacheReplicaEngineCPUUtilization"
    },
    "DisableScaleIn": true
}
```

### Applying a scaling policy to an ElastiCache for Valkey and Redis OSS cluster

After registering your cluster with ElastiCache for Valkey and Redis OSS auto scaling and defining a scaling policy, you apply the scaling policy to the registered cluster. To apply a scaling policy to an ElastiCache for Valkey and Redis OSS cluster, you can use the AWS CLI or the Application Auto Scaling API.

#### Using the AWS CLI

To apply a scaling policy to your ElastiCache for Valkey and Redis OSS cluster, use the put-scaling-policy command with the following parameters:

- --policy-name – The name of the scaling policy.
- --policy-type – Set this value to `TargetTrackingScaling`.
- --resource-id – The resource identifier for the cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- --service-namespace – Set this value to elasticache.
- --scalable-dimension – Set this value to `elasticache:replication-group:Replicas`.
- --target-tracking-scaling-policy-configuration – The target-tracking scaling policy configuration to use for the cluster.

#### Example

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to a cluster named `myscalablecluster` with ElastiCache auto scaling. To do so, you use a policy configuration saved in a file named `config.json`.

For Linux, macOS, or Unix:

```
aws application-autoscaling put-scaling-policy \
```

```
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id replication-group/myscalablecluster \
    --service-namespace elasticache \
    --scalable-dimension elasticache:replication-group:Replicas \
    --target-tracking-scaling-policy-configuration file://config.json
```

```
{"TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {"PredefinedMetricType": "ElastiCacheReplicaEngineCPUUtilization"
    },
    "DisableScaleIn": true
}
```

For Windows:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id replication-group/myscalablecluster ^
    --service-namespace elasticache ^
    --scalable-dimension elasticache:replication-group:Replicas ^
    --target-tracking-scaling-policy-configuration file://config.json
```

**Using the API**

To apply a scaling policy to your ElastiCache cluster with the Application Auto Scaling API, use the PutScalingPolicy Application Auto Scaling API operation with the following parameters:

- PolicyName – The name of the scaling policy.
- PolicyType – Set this value to `TargetTrackingScaling`.
- ResourceID – The resource identifier for the cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the ElastiCache for Redis OSS cluster, for example `replication-group/myscalablecluster`.
- ServiceNamespace – Set this value to elasticache.
- ScalableDimension – Set this value to `elasticache:replication-group:Replicas`.
- TargetTrackingScalingPolicyConfiguration – The target-tracking scaling policy configuration to use for the cluster.

### Example

In the following example, you apply a target-tracking scaling policy named `scalablepolicy` to an cluster named `myscalablecluster` with ElastiCache auto scaling. You use a policy configuration based on the `ElastiCacheReplicaEngineCPUUtilization` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS
{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "elasticache",
    "ResourceId": "replication-group/myscalablecluster",
    "ScalableDimension": "elasticache:replication-group:Replicas",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "ElastiCacheReplicaEngineCPUUtilization"
        }
    }
}
```

### Editing a scaling policy

You can edit a scaling policy using the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

### Editing a scaling policy using the AWS Management Console

You can only edit policies with type Predefined metrics by using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. In the navigation pane, choose **Valkey** or **Redis OSS**
3. Choose the cluster that you want to add a policy to (choose the cluster name and not the button to its left).
4. Choose the **Auto Scaling policies** tab.
5. Under **Scaling policies**, choose the button to the left of the Auto Scaling policy you wish to change, and then choose **Modify**.
6. Make the requisite changes to the policy.
7. Choose **Modify**.
8. Make changes to the policy.
9. Choose **Modify**.

#### Editing a scaling policy using the AWS CLI or the Application Auto Scaling API

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- When using the Application Auto Scaling API, specify the name of the policy you want to edit in the `PolicyName` parameter. Specify new values for the parameters you want to change.

For more information, see Applying a scaling policy to an ElastiCache for Valkey and Redis OSS cluster.

### Deleting a scaling policy

You can delete a scaling policy using the AWS Management Console, the AWS CLI or the Application Auto Scaling API

#### Deleting a scaling policy using the AWS Management Console

You can only edit policies with type Predefined metrics by using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**
3. Choose the cluster whose auto scaling policy you want to delete.
4. Choose the **Auto Scaling policies** tab.

5. Under **Scaling policies**, choose the auto scaling policy, and then choose **Delete**.

### Deleting a scaling policy using the AWS CLI or the Application Auto Scaling API

You can use the AWS CLI or the Application Auto Scaling API to delete a scaling policy from an ElastiCache cluster.

#### CLI

To delete a scaling policy from your ElastiCache for Valkey and Redis OSS cluster, use the delete-scaling-policy command with the following parameters:

- --policy-name – The name of the scaling policy.
- --resource-id – The resource identifier for the cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- --service-namespace – Set this value to elasticache.
- --scalable-dimension – Set this value to `elasticache:replication-group:Replicas`.

#### Example

In the following example, you delete a target-tracking scaling policy named `myscalablepolicy` from an ELC; cluster named `myscalablecluster`.

For Linux, macOS, or Unix:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id replication-group/myscalablecluster \
    --service-namespace elasticache \
    --scalable-dimension elasticache:replication-group:Replicas \
```

For Windows:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id replication-group/myscalablecluster ^
```

```
    --service-namespace elasticache ^
    --scalable-dimension elasticache:replication-group:Replicas ^
```

### API

To delete a scaling policy from your ElastiCache for Valkey and Redis OSS cluster, use the DeleteScalingPolicy Application Auto Scaling API operation with the following parameters:

- PolicyName – The name of the scaling policy.
- ResourceID – The resource identifier for the cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the cluster, for example `replication-group/myscalablecluster`.
- ServiceNamespace – Set this value to elasticache.
- ScalableDimension – Set this value to `elasticache:replication-group:Replicas`.

In the following example, you delete a target-tracking scaling policy named `myscalablepolicy` from a cluster named `myscalablecluster` with the Application Auto Scaling API.

```
POST / HTTP/1.1
>>>>>>> mainline
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS
{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "elasticache",
    "ResourceId": "replication-group/myscalablecluster",
    "ScalableDimension": "elasticache:replication-group:Replicas"
}
```

### Use CloudFormation for Auto Scaling policies

This snippet shows how to create a scheduled action and apply it to an AWS::ElastiCache::ReplicationGroup resource using the AWS::ApplicationAutoScaling::ScalableTarget resource. It uses the Fn::Join and Ref

intrinsic functions to construct the `ResourceId` property with the logical name of the `AWS::ElastiCache::ReplicationGroup` resource that is specified in the same template.

```
ScalingTarget:
   Type: 'AWS::ApplicationAutoScaling::ScalableTarget'
   Properties:
     MaxCapacity: 0
     MinCapacity: 0
     ResourceId: !Sub replication-group/${logicalName}
     ScalableDimension: 'elasticache:replication-group:Replicas'
     ServiceNamespace: elasticache
     RoleARN: !Sub "arn:aws:iam::${AWS::AccountId}:role/aws-
service-role/elasticache.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG"

  ScalingPolicy:
    Type: "AWS::ApplicationAutoScaling::ScalingPolicy"
    Properties:
      ScalingTargetId: !Ref ScalingTarget
      ServiceNamespace: elasticache
      PolicyName: testpolicy
      PolicyType: TargetTrackingScaling
      ScalableDimension: 'elasticache:replication-group:Replicas'
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ElastiCacheReplicaEngineCPUUtilization
        TargetValue: 40
```

**Scheduled scaling**

Scaling based on a schedule enables you to scale your application in response to predictable changes in demand. To use scheduled scaling, you create scheduled actions, which tell ElastiCache for Valkey and Redis OSS to perform scaling activities at specific times. When you create a scheduled action, you specify an existing ElastiCache cluster, when the scaling activity should occur, minimum capacity, and maximum capacity. You can create scheduled actions that scale one time only or that scale on a recurring schedule.

You can only create a scheduled action for ElastiCache clusters that already exist. You can't create a scheduled action at the same time that you create a cluster.

For more information on terminology for scheduled action creation, management, and deletion, see Commonly used commands for scheduled action creation, management, and deletion

**To create a one-time scheduled action:**

Similar to Shard dimension. See Scheduled scaling.

**To delete a scheduled action**

Similar to Shard dimension. See Scheduled scaling.

**To manage scheduled scaling using the AWS CLI**

Use the following application-autoscaling APIs:

- put-scheduled-action
- describe-scheduled-actions
- delete-scheduled-action

### Use CloudFormation to create Auto Scaling policies

This snippet shows how to create a scheduled action and apply it to an AWS::ElastiCache::ReplicationGroup resource using the AWS::ApplicationAutoScaling::ScalableTarget resource. It uses the Fn::Join and Ref intrinsic functions to construct the `ResourceId` property with the logical name of the `AWS::ElastiCache::ReplicationGroup` resource that is specified in the same template.

```
ScalingTarget:
   Type: 'AWS::ApplicationAutoScaling::ScalableTarget'
   Properties:
     MaxCapacity: 0
     MinCapacity: 0
     ResourceId: !Sub replication-group/${logicalName}
     ScalableDimension: 'elasticache:replication-group:Replicas'
     ServiceNamespace: elasticache
     RoleARN: !Sub "arn:aws:iam::${AWS::AccountId}:role/aws-
service-role/elasticache.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG"
     ScheduledActions:
       - EndTime: '2020-12-31T12:00:00.000Z'
         ScalableTargetAction:
           MaxCapacity: '5'
           MinCapacity: '2'
```

```
        ScheduledActionName: First
        Schedule: 'cron(0 18 * * ? *)'
```

## Modifying cluster mode

Valkey and Redis OSS are a distributed in-memory databases that supports sharding and replication. ElastiCache Valkey and Redis OSS clusters are the distributed implementation that allows data to be partitioned across multiple nodes. An ElastiCache for Redis OSS cluster has two modes of operation, Cluster mode enabled (CME) and cluster mode disabled (CMD). In CME, a Valkey and Redis OSS engine works as a distributed database with multiple shards and nodes, while in CMD, Valkey and Redis OSS work as a single node.

Before migrating from CMD to CME, the following conditions must be met:

**Important**

Cluster mode configuration can only be changed from cluster mode disabled to cluster mode enabled. Reverting this configuration is not possible.

- The cluster may only have keys in database 0 only.
- Applications must use a Valkey or Redis OSS client that is capable of using Cluster protocol and use a configuration endpoint.
- Auto-failover must be enabled on the cluster with a minimum of 1 replica.
- The minimum engine version required for migration is Valkey 7.2 and above, or Redis OSS 7.0 and above.

To migrate from CMD to CME, the cluster mode configuration must be changed from cluster mode disabled to cluster mode enabled. This is a two-step procedure that ensures cluster availability during the migration process.

**Note**

You need to provide a parameter group with cluster-enabled configuration, that is, the cluster-enabled parameter is set as `yes`. If you are using a default parameter group, ElastiCache for Redis OSS will automatically pick the corresponding default parameter group with a cluster-enabled configuration. The cluster-enabled parameter value is set to

no for a CMD cluster. As the cluster moves to the compatible mode, the cluster-enabled parameter value is updated to `yes` as part of the modification action.
For more information, see Configuring engine parameters using ElastiCache parameter groups

1. **Prepare** – Create a test CME cluster and make sure your stack is ready to work with it. ElastiCache for Redis OSS has no way to verify your readiness. For more information, see Creating a cluster for Valkey or Redis OSS.
2. **Modify existing CMD Cluster Configuration to cluster mode compatible** – In this mode, there will be a single shard deployed, and ElastiCache for Redis OSS will work as a single node but also as a single shard cluster. Compatible mode means the client application can use either protocol to communicate with the cluster. In this mode, applications must be reconfigured to start using Valkey or Redis OSS Cluster protocol and configuration endpoint. To change the Valkey or Redis OSS cluster mode to cluster mode compatible, follow the steps below:

   **Note**

   In compatible mode, other modification operations such as scaling and engine version are not allowed for the cluster. Additionally, parameters (excluding `cacheParameterGroupName`) cannot be modified when defining cluster-mode parameter within the ModifyReplicationGroup request.

   a. Using the AWS Management Console, see Modifying a replication group and set the cluster mode to **Compatible**

   b. Using the API, see ModifyReplicationGroup and update the `ClusterMode` parameter to `compatible`.

   c. Using the AWS CLI, see modify-replication-group and update the `cluster-mode` parameter to `compatible`.

   After changing the Valkey or Redis OSS cluster mode to cluster mode compatible, the DescribeReplicationGroups API will return the ElastiCache for Redis OSS cluster configuration endpoint. The cluster configuration endpoint is a single endpoint that can be used by applications to connect to the cluster. For more information, see Finding connection endpoints in ElastiCache.

3. **Modify Cluster Configuration to cluster mode enabled** – Once the cluster mode is set to cluster mode compatible, the second step is to modify the cluster configuration to cluster mode enabled. In this mode, a single shard is running, and customers can now scale their clusters or modify other cluster configurations.

   To change the cluster mode to enabled, follow the steps below:

   Before you begin, make sure your Valkey or Redis OSS clients have migrated to using cluster protocol and that the cluster's configuration endpoint is not in use.

   a. Using the AWS Management Console, see Modifying a replication group and set the cluster mode to **Enabled**.

   b. Using the API, see ModifyReplicationGroup and update the `ClusterMode` parameter to `enabled`.

   c. Using the AWS CLI, see modify-replication-group and update the `cluster-mode` parameter to `enabled`.

   After changing the cluster mode to enabled, the endpoints will be configured as per the Valkey or Redis OSS cluster specification. The DescribeReplicationGroups API will return the cluster mode parameter as `enabled` and the cluster endpoints that are now available to be used by applications to connect to the cluster.

   Note that the cluster endpoints will change once the cluster mode is changed to enabled. Make sure to update your applications with the new endpoints.

You can also choose to revert back to cluster mode disabled (CMD) from cluster mode compatible and preserve the original configurations.

**Modify Cluster Configuration to cluster mode disabled from cluster mode compatible**

1. Using the AWS Management Console, see Modifying a replication group and set the cluster mode to **Disabled**
2. Using the API, see ModifyReplicationGroup and update the `ClusterMode` parameter to `disabled`.
3. Using the AWS CLI, see modify-replication-group and update the `cluster-mode` parameter to `disabled`.

After changing the cluster mode to disabled, the DescribeReplicationGroups API will return the cluster mode parameter as `disabled`.

## Replication across AWS Regions using global datastores

> **Note**
>
> Global Datastore is currently available for node-based clusters only.

By using the Global Datastore feature, you can work with fully managed, fast, reliable, and secure Valkey or Redis OSS cluster replication across AWS Regions. Using this feature, you can create cross-Region read replica clusters to enable low-latency reads and disaster recovery across AWS Regions.

In the following sections, you can find a description of how to work with global datastores.

**Topics**


- Overview
- Prerequisites and limitations
- Using global datastores (console)
- Using global datastores (CLI)


### Overview

Each *global datastore* is a collection of one or more clusters that replicate to one another.

A global datastore consists of the following:

- **Primary (active) cluster** – A primary cluster accepts writes that are replicated to all clusters within the global datastore. A primary cluster also accepts read requests.
- **Secondary (passive) cluster** – A secondary cluster only accepts read requests and replicates data updates from a primary cluster. A secondary cluster needs to be in a different AWS Region than the primary cluster.

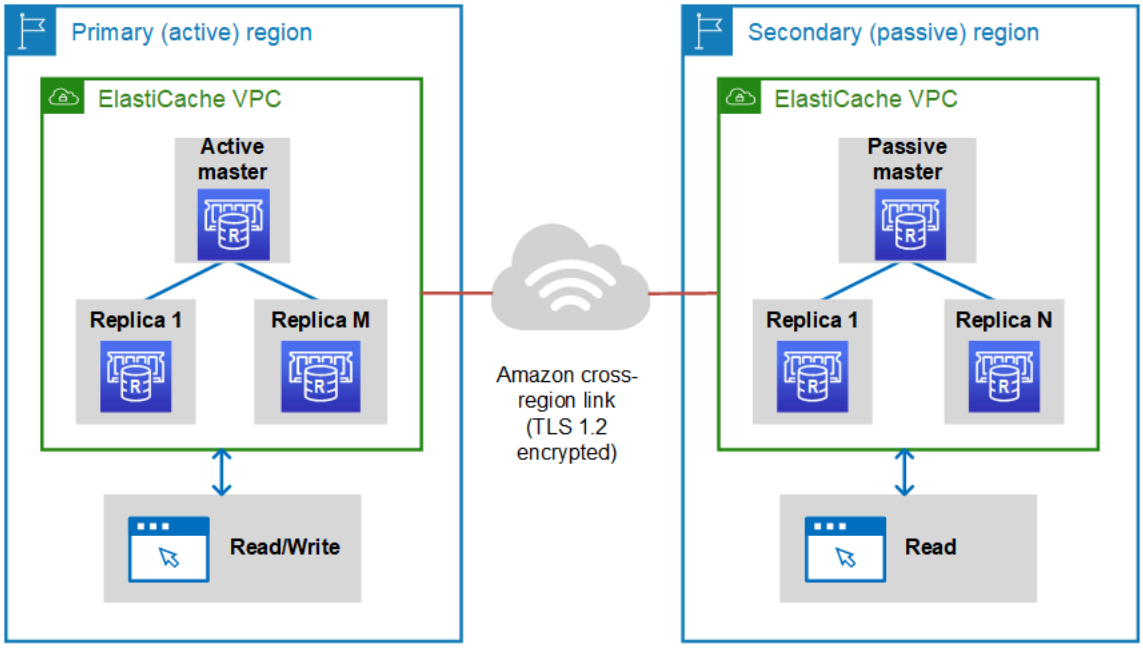
When you create a global datastore in ElastiCache for Valkey or Redis OSS, it automatically replicates your data from the primary cluster to the secondary cluster. You choose the AWS Region

where the Valkey or Redis OSS data should be replicated and then create a secondary cluster in that AWS Region. ElastiCache then sets up and manages automatic, asynchronous replication of data between the two clusters.

Using a global datastore for Valkey or Redis OSS provides the following advantages:

- **Geolocal performance** – By setting up remote replica clusters in additional AWS Regions and synchronizing your data between them, you can reduce latency of data access in that AWS Region. A global datastore can help increase the responsiveness of your application by serving low-latency, geolocal reads across AWS Regions.
- **Disaster recovery** – If your primary cluster in a global datastore experiences degradation, you can promote a secondary cluster as your new primary cluster. You can do so by connecting to any AWS Region that contains a secondary cluster.

The following diagram shows how global datastores can work.




## Prerequisites and limitations

Before getting started with global datastores, be aware of the following:

- Global datastores are supported in the following AWS Regions:
  - **Africa** - Cape Town
  - **Asia Pacific** - Hong Kong, Hyderabad, Jakarta, Malaysia, Melbourne, Mumbai, Osaka, Seoul, Singapore, Sydney, Thailand, and Tokyo

- **Canada** - Canada Central and Canada West (Calgary)
- **China** - Beijing and Ningxia
- **Europe** - Frankfurt, London, Ireland, Milan, Paris, Spain, Stockholm, and Zurich
- **AWS GovCloud** -US-West and US-East
- **Israel** - Tel Aviv
- **Middle East** - Bahrain and UAE
- **US** - East (N. Virginia and Ohio) and US West (N. California and Oregon)
- **South America** - Mexico (Central) and São Paulo

- All clusters—primary and secondary—in your global datastore should have the same number of primary nodes, node type, engine version, and number of shards (in case of cluster-mode enabled). Each cluster in your global datastore can have a different number of read replicas to accommodate the read traffic local to that cluster.

  Replication must be enabled if you plan to use an existing single-node cluster.

  > **Important**
  >
  > When you add a secondary cluster to a global datastore, the secondary cluster inherits the Multi-AZ and automatic failover settings from the primary cluster. Automatic failover requires at least 2 nodes (1 primary and 1 replica). If you enabled automatic failover on the primary cluster, each secondary cluster must also have at least 1 replica. You cannot add a single-node secondary cluster to a global datastore if you enabled automatic failover on the primary.

- Global datastores are supported on the following instance families in size large and above: M5, M6g, M7g, M8g, R5, R6g, R7g, R8g, C7gn, and C8gn. Data tiering node types aren't supported. Previous generation instance types (such as M4 and R4) are not supported.

  > **Note**
  >
  > Supported node types may vary between AWS Regions. For more details, see Amazon ElastiCache pricing.

- You can set up replication for a primary cluster from one AWS Region to a secondary cluster in up to two other AWS Regions.

> **Note**
>
> The exception to this are China (Beijing) Region and China (Ningxia) regions, where replication can only occur between the two regions.

- Global datastores require clusters running in an Amazon VPC. For more information, see Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC.

> **Note**
>
> At this time, you can't use global datastores in Using local zones with ElastiCache.

- ElastiCache doesn't support autofailover from one AWS Region to another. When needed, you can promote a secondary cluster manually. For an example, see Promoting the secondary cluster to primary.
- To bootstrap from existing data, use an existing cluster as primary to create a global datastore. We don't support adding an existing cluster as secondary. The process of adding the cluster as secondary wipes data, which may result in data loss.
- Parameter updates are applied to all clusters when you modify a local parameter group of a cluster belonging to a global datastore.
- You can scale regional clusters both vertically (scaling up and down) and horizontally (scaling in and out). You can scale the clusters by modifying the global datastore. All the regional clusters in the global datastore are then scaled without interruption. For more information, see Scaling ElastiCache.
- Global datastores support encryption at rest, encryption in transit, and AUTH.
- Global datastores doesn't support Internet Protocol version 6 (IPv6).
- Global datastores support AWS KMS keys. For more information, see AWS key management service concepts in the *AWS Key Management Service Developer Guide.*
- Global datastores are not supported with durability-enabled clusters.
- When a cluster is associated with a global datastore, ElastiCache automatically disables the *Auto upgrade minor versions* setting to prevent engine version inconsistencies between member clusters. This setting cannot be re-enabled while the cluster belongs to a global datastore.
- Global Datastore doesn't support cross-account deployments. All primary and secondary clusters must reside within the same AWS account.

**Note**

Global datastores support pub/sub messaging with the following stipulations:

- For cluster-mode disabled, pub/sub is fully supported. Events published on the primary cluster of the primary AWS Region are propagated to secondary AWS Regions.
- For cluster mode enabled, the following applies:
  - For published events that aren't in a keyspace, only subscribers in the same AWS Region receive the events.
  - For published keyspace events, subscribers in all AWS Regions receive the events.

## Using global datastores (console)

To create a global datastore using the console, follow this two-step process:

1. Create a primary cluster, either by using an existing cluster or creating a new cluster. The engine must be Valkey 7.2 or later, or Redis OSS 5.0.6 or later.
2. Add up to two secondary clusters in different AWS Regions, again using Valkey 7.2 or later, or Redis OSS 5.0.6 or later.

The following procedures guide you on how to create a global datastore for Valkey or Redis OSS and perform other operations using the ElastiCache console.

### Topics


- Creating a global datastore using an existing cluster
- Creating a new global datastore using a new primary cluster
- Viewing global datastore details
- Adding a Region to a global datastore
- Modifying a global datastore
- Promoting the secondary cluster to primary
- Removing a Region from a global datastore
- Deleting a global datastore

### Creating a global datastore using an existing cluster

In this scenario, you use an existing cluster to serve as the primary of the new global datastore. You then create a secondary, read-only cluster in a separate AWS Region. This secondary cluster receives automatic and asynchronous updates from the primary cluster.

> **Important**
>
> The existing cluster must use an engine that is Valkey 7.2 or later or Redis OSS 5.0.6 or later.

#### To create a global datastore using an existing cluster

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores** and then choose **Create global datastore**.
3. On the **Primary cluster settings** page, do the following:
   - In the **Global Datastore info** field, enter a name for the new global datastore.
   - (Optional) Enter a **Description** value.
4. Under **Regional cluster**, select **Use existing regional cluster**.
5. Under **Existing cluster**, select the existing cluster you want to use.

   > **Note**
   >
   > Clusters that are already part of a global datastore don't appear in this list. To add a new region to an existing global datastore, see the section called “Adding a Region to a global datastore”.

6. Keep the following options as they are. They're prepopulated to match the primary cluster configuration, you can't change them.
   - Engine version
   - Node type
   - Parameter group

> **Note**
>
> ElastiCache autogenerates a new parameter group from values of the provided parameter group and applies the new parameter group to the cluster. Use this new parameter group to modify parameters on a global datastore. Each autogenerated parameter group is associated with one and only one cluster and, therefore, only one global datastore.

- Number of shards
- Encryption at rest – Enables encryption of data stored on disk. For more information, see Encryption at rest.

> **Note**
>
> You can supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key. For more information, see Using Customer Managed AWS KMS keys.

- Encryption in-transit – Enables encryption of data on the wire. For more information, see Encryption in transit. For Valkey 7.2 and onwards and Redis OSS 6.0 onwards, if you enable encryption in-transit you are prompted to specify one of the following **Access Control** options:
  - **No Access Control** – This is the default setting. This indicates no restrictions.
  - **User Group Access Control List** – Choose a user group with a defined set of users and permissions on available operations. For more information, see Managing User Groups with the Console and CLI.
  - **AUTH Default User** – An authentication mechanism for a Valkey or Redis OSS server. For more information, see AUTH.

7. (Optional) As needed, update the remaining secondary cluster settings. These are prepopulated with the same values as the primary cluster, but you can update them to meet specific requirements for that cluster.

   - Port
   - Number of replicas
   - Subnet group

- Preferred Availability Zone(s)
- Security groups
- Customer Managed (AWS KMS key)
- AUTH Token
- Enable automatic backups
- Backup retention period
- Backup window
- Maintenance window
- Topic for SNS notification

8. Choose **Create**. Doing this sets the status of the global datastore to **Creating**. The status transitions to **Modifying** after the primary cluster is associated to the global datastore and the secondary cluster is in **Associating** status.

   After the primary cluster and secondary clusters are associated with the global datastore, the status changes to **Available**. At this point, you have a primary cluster that accepts reads and writes and secondary clusters that accept reads replicated from the primary cluster.

   The page is updated to indicate whether a cluster is part of a global datastore, including:

   - **Global Datastore** – The name of the global datastore to which the cluster belongs.
   - **Global Datastore Role** – The role of the cluster, either primary or secondary.

You can add up to one additional secondary cluster in a different AWS Region. For more information, see Adding a Region to a global datastore.

**Creating a new global datastore using a new primary cluster**

If you choose to create a global datastore with a new cluster, use the following procedure.

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores** and then choose **Create global datastore**.
3. Under **Primary cluster settings**, do the following:

   a. For **Cluster mode**, choose **Enabled** or **Disabled**.

b. For **Global Datastore info** enter a value for **Name**. ElastiCache uses the suffix to generate a unique name for the global datastore. You can search for the global datastore by using the suffix that you specify here.

c. (Optional) Enter a value for **Global Datastore Description**.

4. Under **Regional cluster**:

a. For **Region**, choose an available AWS Region.

b. Choose **Create new regional cluster** or **Use existing regional cluster**

c. If you choose **Create new regional cluster**, under **Cluster info**, enter a name and optional description of the cluster.

d. Under **Location**, we recommend you accept the default settings for **Multi-AZ** and **Auto-failover**.

5. Under **Cluster settings**

a. For **Engine version**, choose an available version, which is 5.0.6 or later.

b. For **Port**, use the default port, 6379. If you have a reason to use a different port, enter the port number.

c. For **Parameter group**, choose a parameter group or create a new one. Parameter groups control the runtime parameters of your cluster. For more information on parameter groups, see Valkey and Redis OSS parameters and Creating an ElastiCache parameter group.

> **Note**
>
> When you select a parameter group to set the engine configuration values, that parameter group is applied to all clusters in the global datastore. On the **Parameter Groups** page, the yes/no **Global** attribute indicates whether a parameter group is part of a global datastore.

d. For **Node type**, choose the down arrow (▾).

In the **Change node type** dialog box, choose a value for **Instance family** for the node type that you want. Then choose the node type that you want to use for this cluster, and then choose **Save**.

For more information, see Choosing your node size.

If you choose a data tiering node type, ElastiCache automatically enables data tiering. For more information, see Data tiering in ElastiCache.

e. If you are creating a Valkey or Redis OSS (cluster mode disabled) cluster:

For **Number of replicas**, choose the number of replicas that you want for this cluster.

f. If you are creating a Valkey or Redis OSS (cluster mode enabled) cluster:

i. For **Number of shards**, choose the number of shards (partitions/node groups) that you want for this Valkey or Redis OSS (cluster mode enabled) cluster.

You can change the number of shards in your cluster dynamically. For more information, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters.

ii. For **Replicas per shard**, choose the number of read replica nodes that you want in each shard.

The following restrictions exist for Valkey or Redis OSS (cluster mode enabled).

- If you have Multi-AZ enabled, make sure that you have at least one replica per shard.
- The number of replicas is the same for each shard when creating the cluster using the console.
- The number of read replicas per shard is fixed and cannot be changed. If you find you need more or fewer replicas per shard (API/CLI: node group), you must create a new cluster with the new number of replicas. For more information, see Tutorial: Seeding a new node-based cluster with an externally created backup.

6. For **Subnet group settings**, choose the subnet that you want to apply to this cluster. ElastiCache provides a default IPv4 subnet group or you can choose to create a new one. For IPv6, you need to create a subnet group with an IPv6 CIDR block. If you choose **dual stack**, you then must select a Discovery IP type, either IPv6 or IPv4.

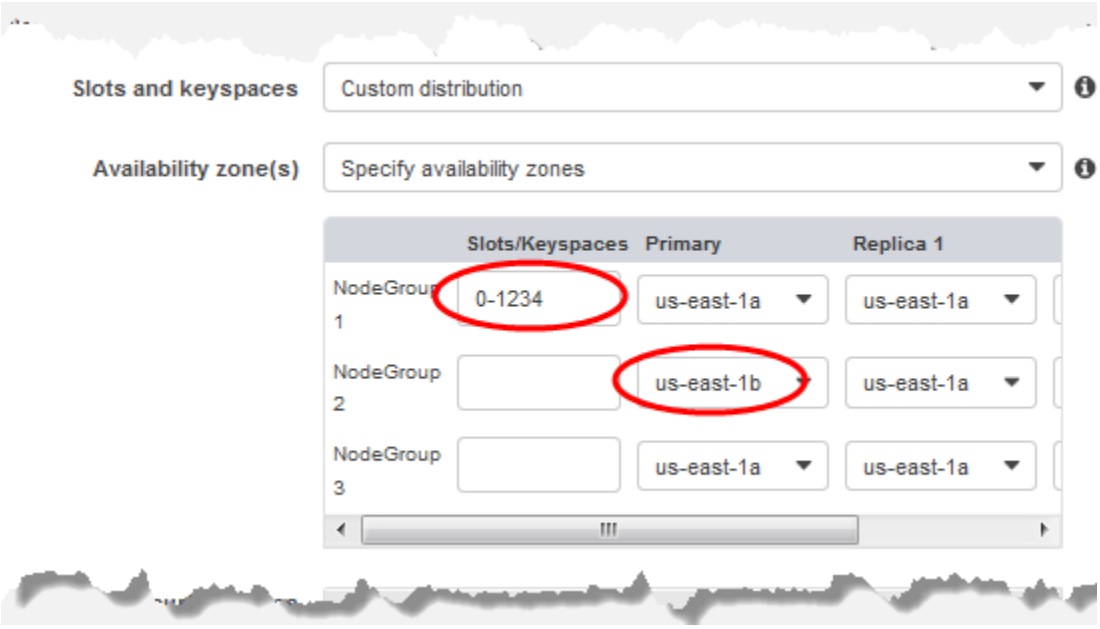
For more information see, Create a subnet in your VPC.

7. For **Availability zone placements**, you have two options:

- **No preference** – ElastiCache chooses the Availability Zone.
- **Specify availability zones** – You specify the Availability Zone for each cluster.

If you chose to specify the Availability Zones, for each cluster in each shard, choose the Availability Zone from the list.

For more information, see Choosing regions and availability zones for ElastiCache.




*Specifying Keyspaces and Availability Zones*

8. Choose **Next**
9. Under **Advanced Valkey and Redis OSS settings**

   - For **Security**:

     i. To encrypt your data, you have the following options:

     - **Encryption at rest** – Enables encryption of data stored on disk. For more information, see Encryption at Rest.

> **Note**
>
> You have the option to supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key. For more information, see Using customer managed keys from AWS KMS.

- **Encryption in-transit** – Enables encryption of data on the wire. For more information, see encryption in transit. For Valkey 7.2 and above and Redis OSS 6.0 and above, if you enable Encryption in-transit you will be prompted to specify one of the following **Access Control** options:
  - **No Access Control** – This is the default setting. This indicates no restrictions on user access to the cluster.
  - **User Group Access Control List** – Select a user group with a defined set of users that can access the cluster. For more information, see Managing User Groups with the Console and CLI.
  - **AUTH Default User** – An authentication mechanism for a Valkey or Redis OSS server. For more information, see AUTH.
- **AUTH** – An authentication mechanism for a Valkey or Redis OSS server. For more information, see AUTH.

> **Note**
>
> RBAC is available on Valkey, and on Redis OSS version 6.0 and later. On earlier Redis OSS versions, AUTH is the sole option.

ii. For **Security groups**, choose the security groups that you want for this cluster. A *security group* acts as a firewall to control network access to your cluster. You can use the default security group for your VPC or create a new one.

For more information on security groups, see Security groups for your VPC in the *Amazon VPC User Guide.*

10. For regularly scheduled automatic backups, select **Enable automatic backups** and then enter the number of days that you want each automatic backup retained before it is automatically deleted. If you don't want regularly scheduled automatic backups, clear the **Enable automatic backups** check box. In either case, you always have the option to create manual backups.

    For more information on backup and restore, see Snapshot and restore.

11. (Optional) Specify a maintenance window. The *maintenance window* is the time, generally an hour in length, each week when ElastiCache schedules system maintenance for your cluster. You can allow ElastiCache to choose the day and time for your maintenance window (*No preference*), or you can choose the day, time, and duration yourself (*Specify maintenance*

*window*). If you choose *Specify maintenance window* from the lists, choose the *Start day*, *Start time*, and *Duration* (in hours) for your maintenance window. All times are UCT times.

For more information, see Managing ElastiCache cluster maintenance.

12. (Optional) For **Logs**:
    - Under **Log format**, choose either **Text** or **JSON**.
    - Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.
    - Under **Log destination**, choose either **Create new** and enter either your CloudWatch Logs log group name or your Firehose stream name, or choose **Select existing** and then choose either your CloudWatch Logs log group name or your Firehose stream name,
13. For **Tags**, to help you manage your clusters and other ElastiCache resources, you can assign your own metadata to each resource in the form of tags. For mor information, see Tagging your ElastiCache resources.
14. Review all your entries and choices, then make any needed corrections. When you're ready, choose **Next**.
15. After you have configured the cluster in the previous steps, you now configure your secondary cluster details..
16. Under **Regional cluster**, choose the AWS Region where th cluster is located.
17. Under **Cluster info**, enter a name and optional description of the cluster.
18. The following options are prepopulated to match the primary cluster configuration and cannot be changed:
    - Location
    - Engine version
    - Instance type
    - Node type
    - Number of shards
    - Parameter group

    > **Note**
    >
    > ElastiCache autogenerates a new parameter group from values of the provided parameter group and applies the new parameter group to the cluster. Use this new parameter group to modify parameters on a global datastore. Each autogenerated

parameter group is associated with one and only one cluster and, therefore, only one global datastore.

- Encryption at rest – Enables encryption of data stored on disk. For more information, see Encryption at rest.

  **Note**

  You can supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key. For more information, see Using Customer Managed AWS KMS keys.

- Encryption in-transit – Enables encryption of data on the wire. For more information, see Encryption in transit. For Valkey 7.2 and above and Redis OSS 6.4 and above, if you enable encryption in-transit you are prompted to specify one of the following **Access Control** options:
  - **No Access Control** – This is the default setting. This indicates no restrictions on user access to the cluster.
  - **User Group Access Control List** – Choose a user group with a defined set of users that can access the cluster. For more information, see Managing User Groups with the Console and CLI.
  - **AUTH Default User** – An authentication mechanism for a Valkey or Redis OSS server. For more information, see AUTH.

**Note**

For Redis OSS versions between 4.0.2, when Encryption in-transit was first supported, and 6.0.4, AUTH is the sole option.

The remaining secondary cluster settings are pre-populated with the same values as the primary cluster, but the following can be updated to meet specific requirements for that cluster:

- Port
- Number of replicas

- Subnet group
- Preferred Availability Zone(s)
- Security groups
- Customer Managed (AWS KMS key)
- AUTH Token
- Enable automatic backups
- Backup retention period
- Backup window
- Maintenance window
- Topic for SNS notification

19. Choose **Create**. This sets the status of the global datastore to **Creating**. After the primary cluster and secondary clusters are associated with the global datastore, the status changes to **Available**. You have a primary cluster that accepts reads and writes and a secondary cluster that accepts reads replicated from the primary cluster.

    The page is also updated to indicate whether a cluster is part of a global datastore, including the following:

    - **Global Datastore** – The name of the global datastore to which the cluster belongs.
    - **Global Datastore Role** – The role of the cluster, either primary or secondary.

You can add up to one additional secondary cluster in a different AWS Region. For more information, see Adding a Region to a global datastore.

**Viewing global datastore details**

You can view the details of existing global datastores and also modify them on the **Global Datastores** page.

**To view global datastore details**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores** and then choose an available global datastore.

You can then examine the following global datastore properties:

- **Global Datastore Name:** The name of the global datastore
- **Description:** A description of the global datastore
- **Status:** Options include:
  - Creating
  - Modifying
  - Available
  - Deleting
  - Primary-Only - This status indicates the global datastore contains only a primary cluster. Either all secondary clusters are deleted or not successfully created.
- **Cluster Mode:** Either enabled or disabled
- **Engine Version:** The Valkey or Redis OSS engine version running the global datastore
- **Instance Node Type:** The node type used for the global datastore
- **Encryption at-rest:** Either enabled or disabled
- **Encryption in-transit:** Either enabled or disabled
- **AUTH:** Either enabled or disabled

You can make the following changes to the global datastore:

- Adding a Region to a global datastore
- Removing a Region from a global datastore
- Promoting the secondary cluster to primary
- Modifying a global datastore

The Global Datastore page also lists the individual clusters that make up the global datastore and the following properties for each:

- **Region** - The AWS Region where the cluster is stored
- **Role** - Either primary or secondary
- **Cluster name** - The name of the cluster
- **Status** - Options include:

- **Associating** - The cluster is in the process of being associated to the global datastore
- **Associated** - The cluster is associated to the global datastore
- **Disassociating** - The process of removing a secondary cluster from the global datastore using the global datastore name. After this, the secondary cluster no longer receives updates from the primary cluster but it remains as a standalone cluster in that AWS Region.
- **Disassociated** - The secondary cluster has been removed from the global datastore and is now a standalone cluster in its AWS Region.

- **Global Datastore Replica lag** – Shows one value per secondary AWS Region in the global datastore. This is the lag between the secondary Region's primary node and the primary Region's primary node. For cluster mode enabled Valkey or Redis OSS, the lag indicates the maximum delay in seconds among the shards.

### Adding a Region to a global datastore

You can add up to one additional AWS Region to an existing global datastore. In this scenario, you are creating a read-only cluster in a separate AWS Region that receives automatic and asynchronous updates from the primary cluster.

**To add an AWS Region to a global datastore**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores**, and then select an existing global datastore.
3. Choose **Add Regional cluster**, and choose the AWS Region where the secondary cluster is to reside.
4. Under **Cluster info**, enter a value for **Name** and, optionally, for **Description** for the cluster.
5. Keep the following options as they are. They're prepopulated to match the primary cluster configuration, and you can't change them.

    - Engine version
    - Instance type
    - Node type
    - Number of shards
    - Parameter group

> **Note**
>
> ElastiCache autogenerates a new parameter group from values of the provided parameter group and applies the new parameter group to the cluster. Use this new parameter group to modify parameters on a global datastore. Each autogenerated parameter group is associated with one and only one cluster and, therefore, only one global datastore.

- Encryption at rest

> **Note**
>
> You can supply a different encryption key by choosing **Customer Managed AWS KMS key** and choosing the key.

- Encryption in transit
- AUTH

6. (Optional) Update the remaining secondary cluster settings. These are prepopulated with the same values as the primary cluster, but you can update them to meet specific requirements for that cluster:

   - Port
   - Number of replicas
   - Subnet group
   - Preferred Availability Zone(s)
   - Security groups
   - Customer Managed AWS KMS key)
   - AUTH Token
   - Enable automatic backups
   - Backup retention period
   - Backup window
   - Maintenance window
   - Topic for SNS notification

7. Choose **Add**.

### Modifying a global datastore

You can modify properties of regional clusters. Only one modify operation can be in progress on a global datastore, with the exception of promoting a secondary cluster to primary. For more information, see Promoting the secondary cluster to primary.

**To modify a global datastore**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores**, and then for **Global Datastore Name**, choose a global datastore.
3. Choose **Modify** and choose among the following options:

   - **Modify description** – Update the description of the global datastore
   - **Modify engine version** – Only Valkey 7.2 and later or Redis OSS 5.0.6 and later are available.
   - **Modify node type** – Scale regional clusters both vertically (scaling up and down) and horizontally (scaling in and out). Options include the R5 and M5 node families. For more information on node types, see Supported node types.
   - **Modify Automatic Failover** – Enable or disable Automatic Failover. When you enable failover and primary nodes in regional clusters shut down unexpectedly, ElastiCache fails over to one of the regional replicas. For more information, see Auto failover.

   For Valkey or Redis OSS clusters with cluster-mode enabled:

   - **Add shards** – Enter the number of shards to add and optionally specify one or more Availability Zones.
   - **Delete shards** – Choose shards to be deleted in each AWS Region.
   - **Rebalance shards** – Rebalance the slot distribution to ensure uniform distribution across existing shards in the cluster.

To modify a global datastore's parameters, modify the parameter group of any member cluster for the global datastore. ElastiCache applies this change to all clusters within that global datastore automatically. To modify the parameter group of that cluster, use the Valkey or Redis OSS console or the ModifyCacheCluster API operation. For more information, see Modifying an ElastiCache

parameter group. When you modify the parameter group of any cluster contained within a global datastore, it is applied to all the clusters within that global datastore.

To reset an entire parameter group or specific parameters, use the ResetCacheParameterGroup API operation.

**Promoting the secondary cluster to primary**

If the primary cluster or AWS Region becomes unavailable or is experiencing performance issues, you can promote a secondary cluster to primary. Promotion is allowed anytime, even if other modifications are in progress. You can also issue multiple promotions in parallel and the global datastore resolves to one primary eventually. If you promote multiple secondary clusters simultaneously, ElastiCache doesn't guarantee which one ultimately resolves to primary.

**To promote a secondary cluster to primary**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores**.
3. Choose the global datastore name to view the details.
4. Choose the **Secondary** cluster.
5. Choose **Promote to primary**.

   You're then prompted to confirm your decision with the following warning: `Promoting a region to primary will make the cluster in this region as read/writable. Are you sure you want to promote the secondary cluster to primary?`

   `The current primary cluster in primary region will become secondary and will stop accepting writes after this operation completes. Please ensure you update your application stack to direct traffic to the new primary region.`
6. Choose **Confirm** if you want to continue the promotion or **Cancel** if you don't.

If you choose to confirm, your global datastore moves to a **Modifying** state and is unavailable until the promotion is complete.

### Removing a Region from a global datastore

You can remove an AWS Region from a global datastore by using the following procedure.

**To remove an AWS Region from a global datastore**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Global Datastores**.
3. Choose a global datastore.
4. Choose the **Region** you want to remove.
5. Choose **Remove region**.

   > **Note**
   >
   > This option is only available for secondary clusters.

   You're then be prompted to confirm your decision with the following warning: `Removing the region will remove your only available cross region replica for the primary cluster. Your primary cluster will no longer be set up for disaster recovery and improved read latency in remote region. Are you sure you want to remove the selected region from the global datastore?`
6. Choose **Confirm** if you want to continue the promotion or **Cancel** if you don't.

If you choose confirm, the AWS Region is removed and the secondary cluster no longer receives replication updates.

### Deleting a global datastore

To delete a global datastore, first remove all secondary clusters. For more information, see Removing a Region from a global datastore. Doing this leaves the global datastore in **primary-only** status.

**To delete a global datastore**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. On the navigation pane, choose **Global Datastores**.
3. Under **Global Datastore Name** choose the global datastore you want to delete and then choose **Delete**.

   You're then be prompted to confirm your decision with the following warning: `Are you sure you want to delete this Global Datastore?`
4. Choose **Delete**.

The global datastore transitions to **Deleting** status.

## Using global datastores (CLI)

You can use the AWS Command Line Interface (AWS CLI) to control multiple AWS services from the command line and automate them through scripts. You can use the AWS CLI for ad hoc (one-time) operations.

### Downloading and configuring the AWS CLI

The AWS CLI runs on Windows, macOS, or Linux. Use the following procedure to download and configure it.

**To download, install, and configure the CLI**

1. Download the AWS CLI on the AWS command line interface webpage.
2. Follow the instructions for Installing the AWS CLI and Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

### Using the AWS CLI with global datastores

Use the following CLI operations to work with global datastores:

- create-global-replication-group

```
aws elasticache create-global-replication-group \
   --global-replication-group-id-suffix my global datastore  \
   --primary-replication-group-id sample-repl-group  \
   --global-replication-group-description an optional description of the global
 datastore
```

Amazon ElastiCache automatically applies a prefix to the global datastore ID when it is created. Each AWS Region has its own prefix. For instance, a global datastore ID created in the US West (N. California) Region begins with "virxk" along with the suffix name that you provide. The suffix, combined with the autogenerated prefix, guarantees uniqueness of the global datastore name across multiple Regions.

The following table lists each AWS Region and its global datastore ID prefix.

| Region Name/Region | Prefix |
|---|---|
| US East (Ohio) Region<br>`us-east-2` | `fpkhr` |
| US East (N. Virginia) Region<br>`us-east-1` | `ldgnf` |
| US West (N. California) Region<br>`us-west-1` | `virxk` |
| US West (Oregon) Region<br>`us-west-2` | `sgaui` |
| Canada (Central) Region<br>`ca-central-1` | `bxodz` |
| Asia Pacific (Mumbai) Region<br>`ap-south-1` | `erpgt` |
| Asia Pacific (Tokyo) Region<br>`ap-northeast-1` | `quwsw` |

| Region Name/Region | Prefix |
| --- | --- |
| Asia Pacific (Seoul) Region<br>ap-northeast-2 | lfqnh |
| Asia Pacific (Osaka) Region<br>ap-northeast-3 | nlapn |
| Asia Pacific (Singapore) Region<br>ap-southeast-1 | vlqxn |
| Asia Pacific (Sydney) Region<br>ap-southeast-2 | vbgxd |
| Europe (Frankfurt) Region<br>eu-central-1 | iudkw |
| Europe (Ireland) Region<br>eu-west-1 | gxeiz |
| Europe (London) Region<br>eu-west-2 | okuqm |
| EU (Paris) Region<br>eu-west-3 | fgjhi |
| South America (São Paulo) Region<br>sa-east-1 | juxlw |
| China (Beijing) Region<br>cn-north-1 | emvgo |

| Region Name/Region | Prefix |
| --- | --- |
| China (Ningxia) Region<br><br>cn-northwest-1 | ckbem |
| Asia Pacific (Hong Kong) Region<br><br>ap-east-1 | knjmp |
| AWS GovCloud (US-West)<br><br>us-gov-west-1 | sgwui |

- create-replication-group – Use this operation to create secondary clusters for a global datastore by supplying the name of the global datastore to the `--global-replication-group-id` parameter.

```
aws elasticache create-replication-group \
  --replication-group-id secondary replication group name \
  --replication-group-description “Replication group description" \
  --global-replication-group-id global datastore name
```

When calling this operation and passing in a `--global-replication-group-id` value, ElastiCache will infer the values from the primary replication group of the global replication group for the following paramaeters. Do not pass in values for these parameters:

"PrimaryClusterId",

"AutomaticFailoverEnabled",

"NumNodeGroups",

"CacheParameterGroupName",

"CacheNodeType",

"Engine",

"EngineVersion",

```
"CacheSecurityGroupNames",

"EnableTransitEncryption",

"AtRestEncryptionEnabled",

"SnapshotArns",

"SnapshotName"
```

- describe-global-replication-groups

```
aws elasticache describe-global-replication-groups \
   --global-replication-group-id my global datastore  \
   --show-member-info an optional parameter that returns a list of the primary and
 secondary clusters that make up the global datastore
```

- modify-global-replication-group

```
aws elasticache modify-global-replication-group \
   --global-replication-group-id my global datastore  \
   --automatic-failover-enabled \
   --cache-node-type node type \
   --cache-parameter-group-name parameter group name \
   --engine-version engine version \
   --–apply-immediately \
   --global-replication-group-description description
```

**Redis to OSS Valkey cross-engine upgrade for ElastiCache GlobalDataStore**

You can upgrade an existing Redis OSS global replication group to Valkey using the Console, API or CLI.

If you have an existing Redis OSS global replication group you can upgrade to Valkey by specifying the new engine and engine version with modify-global-replication-group API.

For Linux, macOS, or Unix:

```
aws elasticache modify-global-replication-group \
   --global-replication-group-id myGlobalReplGroup \
   --engine valkey \
```

```
   --apply-immediately \
   --engine-version 8.0
```

For Windows:

```
aws elasticache modify-global-replication-group ^
   --global-replication-group-id myGlobalReplGroup ^
   --engine valkey ^
   --apply-immediately ^
   --engine-version 8.0
```

If you have a custom cache parameter group applied to the existing Redis OSS global replication group you wish to upgrade, you will need to pass a custom Valkey cache parameter group in the request as well. The input Valkey custom parameter group must have the same Redis OSS static parameter values as the existing Redis OSS custom parameter group.

For Linux, macOS, or Unix:

```
aws elasticache modify-global-replication-group \
   --global-replication-group-id myGlobalReplGroup \
   --engine valkey \
   --engine-version 8.0 \
   --apply-immediately \
   --cache-parameter-group-name myParamGroup
```

For Windows:

```
aws elasticache modify-global-replication-group ^
   --global-replication-group-id myGlobalReplGroup ^
   --engine valkey ^
   --engine-version 8.0 ^
   --apply-immediately ^
   --cache-parameter-group-name myParamGroup
```

- delete-global-replication-group

```
aws elasticache delete-global-replication-group \
   --global-replication-group-id my global datastore  \
   --retain-primary-replication-group defaults to true
```

- disassociate-global-replication-group

```
aws elasticache disassociate-global-replication-group \
   --global-replication-group-id my global datastore  \
   --replication-group-id my secondary cluster  \
   --replication-group-region the AWS Region in which the secondary cluster resides
```

- failover-global-replication-group

```
aws elasticache failover-replication-group \
   --global-replication-group-id my global datastore \
   --primary-region The AWS Region of the primary cluster \
   --primary-replication-group-id  The name of the global datastore, including the
 suffix.
```

- increase-node-groups-in-global-replication-group

```
aws elasticache increase-node-groups-in-global-replication-group \
   --apply-immediately \
   --global-replication-group-id global-replication-group-name \
   --node-group-count 3
```

- decrease-node-groups-in-global-replication-group

```
aws elasticache decrease-node-groups-in-global-replication-group \
   --apply-immediately \
   --global-replication-group-id global-replication-group-name \
   --node-group-count 3
```

- rebalance-slots-in-global-replication-group

```
aws elasticache rebalance-slots-in-global-replication-group \
   --apply-immediately \
   --global-replication-group-id global-replication-group-name
```

Use help to list all available commands for ElastiCache for Valkey or Redis OSS.

```
aws elasticache help
```

You can also use help to describe a specific command and learn more about its usage:

```
aws elasticache create-global-replication-group help
```

## High availability using replication groups

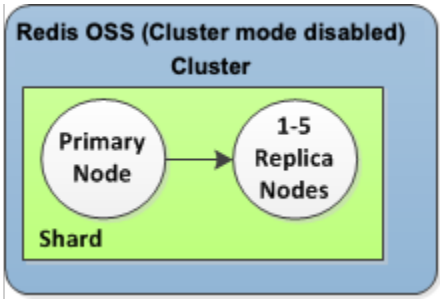
Single-node Amazon ElastiCache Valkey and Redis OSS clusters are in-memory entities with limited data protection services (AOF). If your cluster fails for any reason, you lose all the cluster's data. However, if you're running a Valkey or Redis OSS engine, you can group 2 to 6 nodes into a cluster with replicas where 1 to 5 read-only nodes contain replicate data of the group's single read/write primary node. In this scenario, if one node fails for any reason, you do not lose all your data since it is replicated in one or more other nodes. Due to replication latency, some data may be lost if it is the primary read/write node that fails.

As seen in the following graphic, the replication structure is contained within a shard (called *node group* in the API/CLI) which is contained within a Valkey or Redis OSS cluster. Valkey or Redis OSS (cluster mode disabled) clusters always have one shard. Valkey or Redis OSS (cluster mode enabled) clusters can have up to 500 shards with the cluster's data partitioned across the shards. You can create a cluster with higher number of shards and lower number of replicas totaling up to 90 nodes per cluster. This cluster configuration can range from 90 shards and 0 replicas to 15 shards and 5 replicas, which is the maximum number of replicas allowed.

The node or shard limit can be increased to a maximum of 500 per cluster with ElastiCache for Valkey, and with ElastiCache version 5.0.6 or higher for Redis OSS. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group.

For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.



*Valkey or Redis OSS (cluster mode disabled) cluster has one shard and 0 to 5 replica nodes*

If the cluster with replicas has Multi-AZ enabled and the primary node fails, the primary fails over to a read replica. Because the data is updated on the replica nodes asynchronously, there may be some data loss due to latency in updating the replica nodes. For more information, see Mitigating Failures when Running Valkey or Redis OSS.

> **Note**
>
> For clusters with durability enabled, data is persisted in a Multi-AZ transactional log and can be recovered even if all nodes fail. With synchronous writes, no acknowledged data is lost during failover. With asynchronous writes, up to 10 seconds of data may be lost in case of a failure.

**Topics**


- Understanding Valkey and Redis OSS replication
- Replication with durability enabled
- Replication: Valkey and Redis OSS Cluster Mode Disabled vs. Enabled
- Minimizing downtime in ElastiCache by using Multi-AZ with Valkey and Redis OSS
- Synchronization and backup with durability
- Creating a Valkey or Redis OSS replication group
- Viewing a replication group's details
- Finding replication group endpoints
- Modifying a replication group
- Deleting a replication group
- Changing the number of replicas
- Promoting a read replica to primary, for Valkey or Redis OSS (cluster mode disabled) replication groups

## Understanding Valkey and Redis OSS replication

Redis OSS implements replication in two ways:

- With a single shard that contains all of the cluster's data in each node—Valkey or Redis OSS (cluster mode disabled)
- With data partitioned across up to 500 shards—Valkey or Redis OSS (cluster mode enabled)

Each shard in a replication group has a single read/write primary node and up to 5 read-only replica nodes. You can create a cluster with higher number of shards and lower number of replicas totaling up to 90 nodes per cluster. This cluster configuration can range from 90 shards and 0 replicas to 15 shards and 5 replicas, which is the maximum number of replicas allowed.

The node or shard limit can be increased to a maximum of 500 per cluster if the Redis OSS engine version is 5.0.6 or higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group.

For versions below 5.0.6, the limit is 250 per cluster.

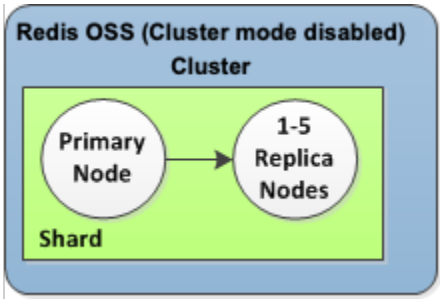
To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

**Topics**


- Valkey or Redis OSS (Cluster Mode Disabled)
- Valkey or Redis OSS (cluster mode enabled)


### Valkey or Redis OSS (Cluster Mode Disabled)

A Valkey or Redis OSS (cluster mode disabled) cluster has a single shard, inside of which is a collection of nodes; one primary read/write node and up to five secondary, read-only replica nodes. Each read replica maintains a copy of the data from the cluster's primary node. Asynchronous replication mechanisms are used to keep the read replicas synchronized with the primary. Applications can read from any node in the cluster. Applications can write only to the primary node. Read replicas improve read throughput and guard against data loss in cases of a node failure.




*Valkey or Redis OSS (cluster mode disabled) cluster with a single shard and replica nodes*

You can use Valkey or Redis OSS (cluster mode disabled) clusters with replica nodes to scale your solution for ElastiCache to handle applications that are read-intensive or to support large numbers of clients that simultaneously read from the same cluster.

All of the nodes in a Valkey or Redis OSS (cluster mode disabled) cluster must reside in the same region.

When you add a read replica to a cluster, all of the data from the primary is copied to the new node. From that point on, whenever data is written to the primary, the changes are asynchronously propagated to all the read replicas.

To improve fault tolerance and reduce write downtime, enable Multi-AZ with Automatic Failover for your Valkey or Redis OSS (cluster mode disabled) cluster with replicas. For more information, see Minimizing downtime in ElastiCache by using Multi-AZ with Valkey and Redis OSS.

> **Note**
>
> Durability is not supported for cluster mode disabled (CMD) clusters.

You can change the roles of the nodes within the Valkey or Redis OSS (cluster mode disabled) cluster, with the primary and one of the replicas exchanging roles. You might decide to do this for performance tuning reasons. For example, with a web application that has heavy write activity, you can choose the node that has the lowest network latency. For more information, see Promoting a read replica to primary, for Valkey or Redis OSS (cluster mode disabled) replication groups.

**Valkey or Redis OSS (cluster mode enabled)**

A Valkey or Redis OSS (cluster mode enabled) cluster is comprised of from 1 to 500 shards (API/CLI: node groups). Each shard has a primary node and up to five read-only replica nodes. The

configuration can range from 90 shards and 0 replicas to 15 shards and 5 replicas, which is the maximum number of replicas allowed.

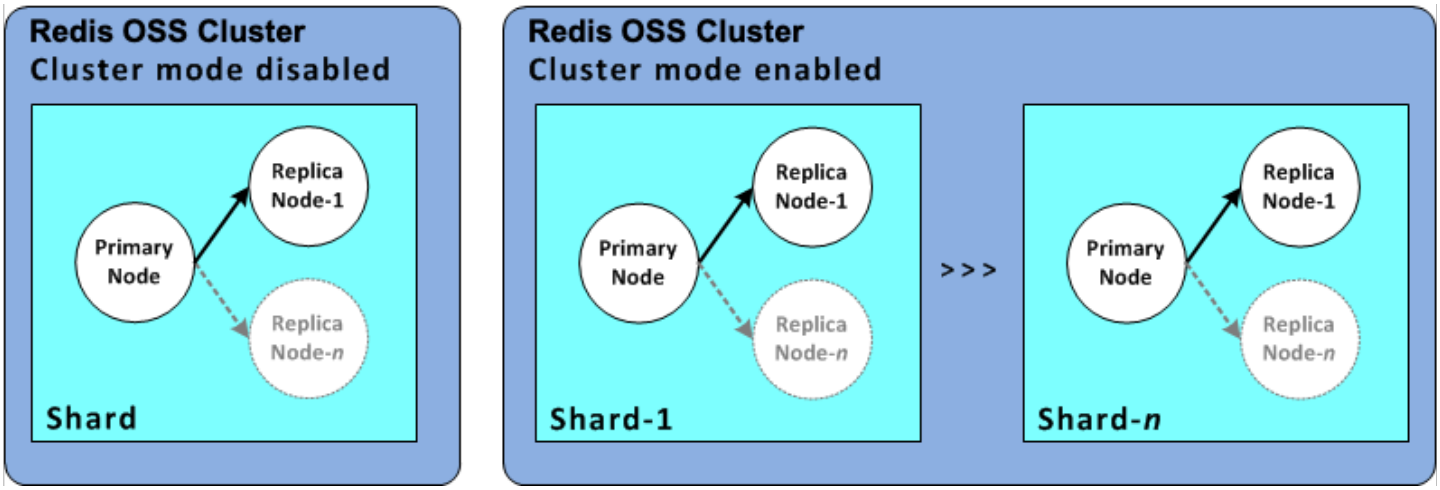
The node or shard limit can be increased to a maximum of 500 per cluster if the engine version is Valkey 7.2 and higher, or Redis OSS 5.0.6 and higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group.

For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

Each read replica in a shard maintains a copy of the data from the shard's primary. Asynchronous replication mechanisms are used to keep the read replicas synchronized with the primary. Applications can read from any node in the cluster. Applications can write only to the primary nodes. Read replicas enhance read scalability and guard against data loss. Data is partitioned across the shards in a Valkey or Redis OSS (cluster mode enabled) cluster.

Applications use the Valkey or Redis OSS (cluster mode enabled) cluster's *configuration endpoint* to connect with the nodes in the cluster. For more information, see Finding connection endpoints in ElastiCache.




*Valkey or Redis OSS (cluster mode enabled) cluster with multiple shards and replica nodes*

All of the nodes in a Valkey or Redis OSS (cluster mode enabled) cluster must reside in the same region. To improve fault tolerance, you can provision both primaries and read replicas in multiple Availability Zones within that region.

Currently, Valkey or Redis OSS (cluster mode enabled) features have some limitations.

> **Note**
>
> For Valkey 9.0+ cluster mode enabled (CME) clusters, durability changes the replication model. Replicas consume writes from the Multi-AZ transactional log rather than streaming directly from the primary.

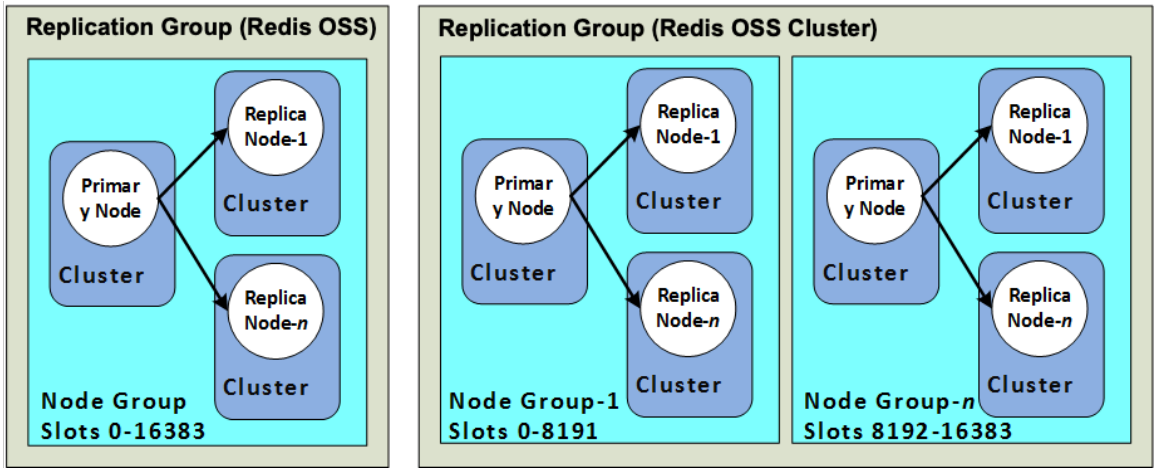
- You cannot manually promote any of the replica nodes to primary.

## Replication with durability enabled

For Valkey 9.0+ clusters with durability enabled, replication is mediated through the Multi-AZ transactional log rather than direct primary-to-replica streaming. The primary node writes to the transactional log, and replicas independently consume committed writes from the log. This architecture means replicas recover independently without imposing load on the primary node.

## Replication: Valkey and Redis OSS Cluster Mode Disabled vs. Enabled

With Valkey and Redis OSS, you have the ability to create one of two distinct types of clusters (API/CLI: replication groups). A Valkey or Redis OSS (cluster mode disabled) cluster always has a single shard (API/CLI: node group) with up to 5 read replica nodes. A Valkey or Redis OSS (cluster mode enabled) cluster has up to 500 shards with 1 to 5 read replica nodes in each.



*Valkey or Redis OSS (cluster mode disabled), and Valkey or Redis OSS (cluster mode enabled) clusters*

The following table summarizes important differences between Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled) clusters.

**Comparing Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled) Clusters**

| Feature | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|
| Modifiable | Yes. Supports adding and deleting replica nodes, and scaling up node type. | Limited. For more information, see Version Management for ElastiCache and Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters. |
| Data Partitioning | No | Yes |
| Shards | 1 | 1 to 500 |
| Read replicas | 0 to 5<br><br>**Important**<br>If you have no replicas and the node fails, you experience total data loss. | 0 to 5 per shard.<br><br>**Important**<br>If you have no replicas and a node fails, you experience loss of all data in that shard. |
| Multi-AZ | Yes, with at least 1 replica.<br><br>Optional. On by default. | Yes<br><br>Optional. On by default. |
| Snapshots (Backups) | Yes, creating a single .rdb file. | Yes, creating a unique .rdb file for each shard. |
| Restore | Yes, using a single .rdb file from a Valkey or Redis OSS | Yes, using .rdb files from either a Valkey or Redis OSS (cluster mode disabled) or a |

| Feature | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|
| | (cluster mode disabled) cluster. | Valkey or Redis OSS (cluster mode enabled) cluster. |
| Supported by | All Valkey and Redis OSS versions | All Valkey and Redis OSS versions |
| Engine upgradeable | Yes, with some limits. For more information, see Version Management for ElastiCache. | Yes, with some limits. For more information, see Version Management for ElastiCache. |
| Encryption | All Valkey versions, and Redis OSS 4.0.10 and later. | All Valkey versions, and Redis OSS 4.0.10 and later. |
| HIPAA Eligible | All Valkey versions, and Redis OSS 4.0.10 and later. | All Valkey versions, and Redis OSS 4.0.10 and later. |
| PCI DSS Compliant | All Valkey versions, and Redis OSS 4.0.10 and later. | All Valkey versions, and Redis OSS 4.0.10 and later. |
| Online resharding | N/A | All Valkey versions, and Redis OSS 4.0.10 and later. |

**Which should I choose?**

When choosing between Valkey or Redis OSS (cluster mode disabled) or Valkey or Redis OSS (cluster mode enabled), consider the following factors:

- **Scaling v. partitioning** – Business needs change. You need to either provision for peak demand or scale as demand changes. Valkey or Redis OSS (cluster mode disabled) supports scaling. You can scale read capacity by adding or deleting replica nodes, or you can scale capacity by scaling up to a larger node type. Both of these operations take time. For more information, see Scaling replica nodes for Valkey or Redis OSS (Cluster Mode Disabled).

Valkey or Redis OSS (cluster mode enabled) supports partitioning your data across up to 500 node groups. You can dynamically change the number of shards as your business needs change. One advantage of partitioning is that you spread your load over a greater number of endpoints, which reduces access bottlenecks during peak demand. Additionally, you can accommodate a larger data set since the data can be spread across multiple servers. For information on scaling your partitions, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters.

- **Node size v. number of nodes** – Because a Valkey or Redis OSS (cluster mode disabled) cluster has only one shard, the node type must be large enough to accommodate all the cluster's data plus necessary overhead. On the other hand, because you can partition your data across several shards when using a Valkey or Redis OSS (cluster mode enabled) cluster, the node types can be smaller, though you need more of them. For more information, see Choosing your node size.

- **Reads v. writes** – If the primary load on your cluster is applications reading data, you can scale a Valkey or Redis OSS (cluster mode disabled) cluster by adding and deleting read replicas. However, there is a maximum of 5 read replicas. If the load on your cluster is write-heavy, you can benefit from the additional write endpoints of a Valkey or Redis OSS (cluster mode enabled) cluster with multiple shards.

Whichever type of cluster you choose to implement, be sure to choose a node type that is adequate for your current and future needs.

## Minimizing downtime in ElastiCache by using Multi-AZ with Valkey and Redis OSS

There are a number of instances where ElastiCache for Valkey and Redis OSS may need to replace a primary node; these include certain types of planned maintenance and the unlikely event of a primary node or Availability Zone failure.

This replacement results in some downtime for the cluster, but if Multi-AZ is enabled, the downtime is minimized. The role of primary node will automatically fail over to one of the read replicas. There is no need to create and provision a new primary node, because ElastiCache will handle this transparently. This failover and replica promotion ensure that you can resume writing to the new primary as soon as promotion is complete.

ElastiCache also propagates the Domain Name Service (DNS) name of the promoted replica. It does so because then if your application is writing to the primary endpoint, no endpoint change is required in your application. If you are reading from individual endpoints, make sure that you change the read endpoint of the replica promoted to primary to the new replica's endpoint.

In case of planned node replacements initiated due to maintenance updates or self-service updates, be aware of the following:

- For Valkey and Redis OSS clusters with cluster mode enabled, the planned node replacements complete while the cluster serves incoming write requests.
- For Valkey and Redis OSS cluster mode disabled clusters with Multi-AZ enabled that run on the 5.0.6 or later engine, the planned node replacements complete while the cluster serves incoming write requests.
- For Valkey and Redis OSS cluster mode disabled clusters with Multi-AZ enabled that run on the 4.0.10 or earlier engine, you might notice a brief write interruption associated with DNS updates. This interruption might take up to a few seconds. This process is much faster than recreating and provisioning a new primary, which is what occurs if you don't enable Multi-AZ.

You can enable Multi-AZ using the ElastiCache Management Console, the AWS CLI, or the ElastiCache API.

> **Note**
>
> **Durability-enabled clusters:** For Valkey 9.0+ clusters with durability enabled, failover behavior differs from the scenarios described below. With durability, all committed data is persisted in a Multi-AZ transactional log. During failovers, synchronous writes ensure

zero data loss, while asynchronous writes may lose up to 10 seconds of uncommitted data. Failed nodes restore all committed data from the transactional log without requiring a full synchronization from the primary. For more information, see the section called “Durability”.

Enabling ElastiCache Multi-AZ on your Valkey or Redis OSS cluster (in the API and CLI, replication group) improves your fault tolerance. This is true particularly in cases where your cluster's read/write primary cluster becomes unreachable or fails for any reason. Multi-AZ is only supported on Valkey and Redis OSS clusters with more than one node in each shard.

**Topics**


- Automatic failover and Multi-AZ
- Enabling Multi-AZ
- Failure scenarios with Multi-AZ responses
- Testing automatic failover
- Limitations on Multi-AZ


### Automatic failover and Multi-AZ

Multi-AZ is the recommended configuration for high availability in ElastiCache. When you enable Multi-AZ, ElastiCache distributes your replicas across different Availability Zones and enables automatic failover. If the primary node fails, ElastiCache automatically promotes the replica with the least replication lag to primary. This provides both AZ-level resilience and rapid recovery from node failures with minimal write interruption.

We recommend enabling Multi-AZ as a single configuration step for most use cases. Multi-AZ requires at least one replica in a different Availability Zone from the primary.

#### How automatic failover works

Automatic failover is the mechanism that Multi-AZ uses to recover from primary node failures. When Multi-AZ is enabled and the primary fails, ElastiCache promotes the replica with the least replication lag to primary. Writes can resume within seconds. ElastiCache then provisions a replacement replica in the Availability Zone where the failed primary was located.

#### What happens without Multi-AZ

If Multi-AZ is not enabled and a primary node fails, ElastiCache takes the primary offline and provisions a new primary node. If replicas exist, ElastiCache syncs the new primary from a replica. If no replicas exist, the new primary starts empty and data is lost unless you restore from a snapshot. This process results in a longer write interruption compared to automatic failover.

> **Note**
>
> While it is possible to have automatic failover enabled without Multi-AZ, replicas are not guaranteed to be in different Availability Zones in this configuration. Failover still promotes a replica to primary if one is available, but this does not protect against AZ-level failures. For most use cases, we recommend enabling Multi-AZ.

Consider the following differences between cluster modes:

- **Valkey or Redis OSS (cluster mode enabled)** – Multi-AZ is enabled by default. Automatic failover is always required and cannot be disabled.
- **Valkey or Redis OSS (cluster mode disabled)** – We recommend enabling Multi-AZ and configuring at least one replica for high availability. You also have the option to enable or disable automatic failover independently.

### Enabling Multi-AZ

You can enable Multi-AZ when you create or modify a cluster (API or CLI, replication group) using the ElastiCache console, AWS CLI, or the ElastiCache API.

You can enable Multi-AZ only on Valkey or Redis OSS (cluster mode disabled) clusters that have at least one available read replica. Clusters without read replicas do not provide high availability or fault tolerance. For information about creating a cluster with replication, see Creating a Valkey or Redis OSS replication group. For information about adding a read replica to a cluster with replication, see Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled).

### Topics

- Enabling Multi-AZ (Console)
- Enabling Multi-AZ (AWS CLI)
- Enabling Multi-AZ (ElastiCache API)

### Enabling Multi-AZ (Console)

You can enable Multi-AZ using the ElastiCache console when you create a new Valkey or Redis OSS cluster or by modifying an existing cluster with replication.

Multi-AZ is enabled by default on Valkey or Redis OSS (cluster mode enabled) clusters.

> **Important**
>
> ElastiCache will automatically enable Multi-AZ only if the cluster contains at least one replica in a different Availability Zone from the primary in all shards.

#### Enabling Multi-AZ when creating a cluster using the ElastiCache console

For more information on this process, see Creating a Valkey (cluster mode disabled) cluster (Console). Be sure to have one or more replicas and enable Multi-AZ.

#### Enabling Multi-AZ on an existing cluster (Console)

For more information on this process, see Modifying a Cluster Using the ElastiCache AWS Management Console.

#### Enabling Multi-AZ (AWS CLI)

The following code example uses the AWS CLI to enable Multi-AZ for the replication group `redis12`.

> **Important**
>
> The replication group `redis12` must already exist and have at least one available read replica.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id redis12 \
    --automatic-failover-enabled \
    --multi-az-enabled \
```

```
    --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id redis12 ^
    --automatic-failover-enabled ^
    --multi-az-enabled ^
    --apply-immediately
```

The JSON output from this command should look something like the following.

```
{
    "ReplicationGroup": {
        "Status": "modifying",
        "Description": "One shard, two nodes",
        "NodeGroups": [
            {
                "Status": "modifying",
                "NodeGroupMembers": [
                    {
                        "CurrentRole": "primary",
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address":
 "redis12-001.v5r9dc.0001.usw2.cache.amazonaws.com"
                        },
                        "CacheClusterId": "redis12-001"
                    },
                    {
                        "CurrentRole": "replica",
                        "PreferredAvailabilityZone": "us-west-2a",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address":
 "redis12-002.v5r9dc.0001.usw2.cache.amazonaws.com"
                        },
                        "CacheClusterId": "redis12-002"
                    }
                ],
```

```
                "NodeGroupId": "0001",
                "PrimaryEndpoint": {
                    "Port": 6379,
                    "Address": "redis12.v5r9dc.ng.0001.usw2.cache.amazonaws.com"
                }
            }
        ],
        "ReplicationGroupId": "redis12",
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "enabling",
        "MultiAZ": "enabled",
        "SnapshotWindow": "07:00-08:00",
        "SnapshottingClusterId": "redis12-002",
        "MemberClusters": [
            "redis12-001",
            "redis12-002"
        ],
        "PendingModifiedValues": {}
    }
}
```

For more information, see these topics in the *AWS CLI Command Reference*:

- create-cache-cluster
- create-replication-group
- modify-replication-group in the *AWS CLI Command Reference.*

### Enabling Multi-AZ (ElastiCache API)

The following code example uses the ElastiCache API to enable Multi-AZ for the replication group `redis12`.

> **Note**
>
> To use this example, the replication group `redis12` must already exist and have at least one available read replica.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyReplicationGroup
```

```
    &ApplyImmediately=true
    &AutoFailover=true
    &MultiAZEnabled=true
    &ReplicationGroupId=redis12
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20140401T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see these topics in the *ElastiCache API Reference*:

- CreateCacheCluster
- CreateReplicationGroup
- ModifyReplicationGroup

### Failure scenarios with Multi-AZ responses

Before the introduction of Multi-AZ, ElastiCache detected and replaced a cluster's failed nodes by recreating and reprovisioning the failed node. If you enable Multi-AZ, a failed primary node fails over to the replica with the least replication lag. The selected replica is automatically promoted to primary, which is much faster than creating and reprovisioning a new primary node. This process usually takes just a few seconds until you can write to the cluster again.

When Multi-AZ is enabled, ElastiCache continually monitors the state of the primary node. If the primary node fails, one of the following actions is performed depending on the type of failure.

**Topics**


- Failure scenarios when only the primary node fails
- Failure scenarios when the primary node and some read replicas fail
- Failure scenarios when the entire cluster fails


### Failure scenarios when only the primary node fails

If only the primary node fails, the read replica with the least replication lag is promoted to primary. A replacement read replica is then created and provisioned in the same Availability Zone as the failed primary.

When only the primary node fails, ElastiCache Multi-AZ does the following:

1. The failed primary node is taken offline.
2. The read replica with the least replication lag is promoted to primary.

   Writes can resume as soon as the promotion process is complete, typically just a few seconds. If your application is writing to the primary endpoint, you don't need to change the endpoint for writes or reads. ElastiCache propagates the DNS name of the promoted replica.
3. A replacement read replica is launched and provisioned.

   The replacement read replica is launched in the Availability Zone that the failed primary node was in so that the distribution of nodes is maintained.
4. The replicas sync with the new primary node.

After the new replica is available, be aware of these effects:

- **Primary endpoint** – You don't need to make any changes to your application, because the DNS name of the new primary node is propagated to the primary endpoint.
- **Read endpoint** – The reader endpoint is automatically updated to point to the new replica nodes.

For information about finding the endpoints of a cluster, see the following topics:

- Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding the Endpoints for Valkey or Redis OSS Replication Groups (AWS CLI)
- Finding Endpoints for Valkey or Redis OSS Replication Groups (ElastiCache API)

**Failure scenarios when the primary node and some read replicas fail**

If the primary and at least one read replica fails, the available replica with the least replication lag is promoted to primary cluster. New read replicas are also created and provisioned in the same Availability Zones as the failed nodes and replica that was promoted to primary.

When the primary node and some read replicas fail, ElastiCache Multi-AZ does the following:

1. The failed primary node and failed read replicas are taken offline.
2. The available replica with the least replication lag is promoted to primary node.

   Writes can resume as soon as the promotion process is complete, typically just a few seconds. If your application is writing to the primary endpoint, there is no need to change the endpoint for writes. ElastiCache propagates the DNS name of the promoted replica.
3. Replacement replicas are created and provisioned.

   The replacement replicas are created in the Availability Zones of the failed nodes so that the distribution of nodes is maintained.
4. All clusters sync with the new primary node.

Make the following changes to your application after the new nodes are available:

- **Primary endpoint** – Don't make any changes to your application. The DNS name of the new primary node is propagated to the primary endpoint.

- **Read endpoint** – The read endpoint is automatically updated to point to the new replica nodes.

For information about finding the endpoints of a replication group, see the following topics:

- Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding the Endpoints for Valkey or Redis OSS Replication Groups (AWS CLI)
- Finding Endpoints for Valkey or Redis OSS Replication Groups (ElastiCache API)

**Failure scenarios when the entire cluster fails**

If everything fails, all the nodes are recreated and provisioned in the same Availability Zones as the original nodes.

In this scenario, all the data in the cluster is lost due to the failure of every node in the cluster. This occurrence is rare.

When the entire cluster fails, ElastiCache Multi-AZ does the following:

1. The failed primary node and read replicas are taken offline.
2. A replacement primary node is created and provisioned.
3. Replacement replicas are created and provisioned.

   The replacements are created in the Availability Zones of the failed nodes so that the distribution of nodes is maintained.

   Because the entire cluster failed, data is lost and all the new nodes start cold.

Because each of the replacement nodes has the same endpoint as the node it's replacing, you don't need to make any endpoint changes in your application.

For information about finding the endpoints of a replication group, see the following topics:

- Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding the Endpoints for Valkey or Redis OSS Replication Groups (AWS CLI)
- Finding Endpoints for Valkey or Redis OSS Replication Groups (ElastiCache API)

We recommend that you create the primary node and read replicas in different Availability Zones to raise your fault tolerance level.

### Testing automatic failover

After you enable automatic failover, you can test it using the ElastiCache console, the AWS CLI, and the ElastiCache API.

When testing, note the following:

- You can use this operation to test automatic failover on up to 15 shards (called node groups in the ElastiCache API and AWS CLI) in any rolling 24-hour period.
- If you call this operation on shards in different clusters (called replication groups in the API and CLI), you can make the calls concurrently.
- In some cases, you might call this operation multiple times on different shards in the same Valkey or Redis OSS (cluster mode enabled) replication group. In such cases, the first node replacement must complete before a subsequent call can be made.
- To determine whether the node replacement is complete, check events using the Amazon ElastiCache console, the AWS CLI, or the ElastiCache API. Look for the following events related to automatic failover, listed here in order of likely occurrence:

  1. Replication group message: `Test Failover API called for node group <node-group-id>`
  2. Cache cluster message: `Failover from primary node <primary-node-id> to replica node <node-id> completed`
  3. Replication group message: `Failover from primary node <primary-node-id> to replica node <node-id> completed`
  4. Cache cluster message: `Recovering cache nodes <node-id>`
  5. Cache cluster message: `Finished recovery for cache nodes <node-id>`

  For more information, see the following:

  - Viewing ElastiCache events in the *ElastiCache User Guide*
  - DescribeEvents in the *ElastiCache API Reference*
  - describe-events in the *AWS CLI Command Reference.*
- This API is designed for testing the behavior of your application in case of ElastiCache failover. It is not designed to be an operational tool for initiating a failover to address an issue with the cluster. Moreover, in certain conditions such as large-scale operational events, AWS may block this API.

**Topics**

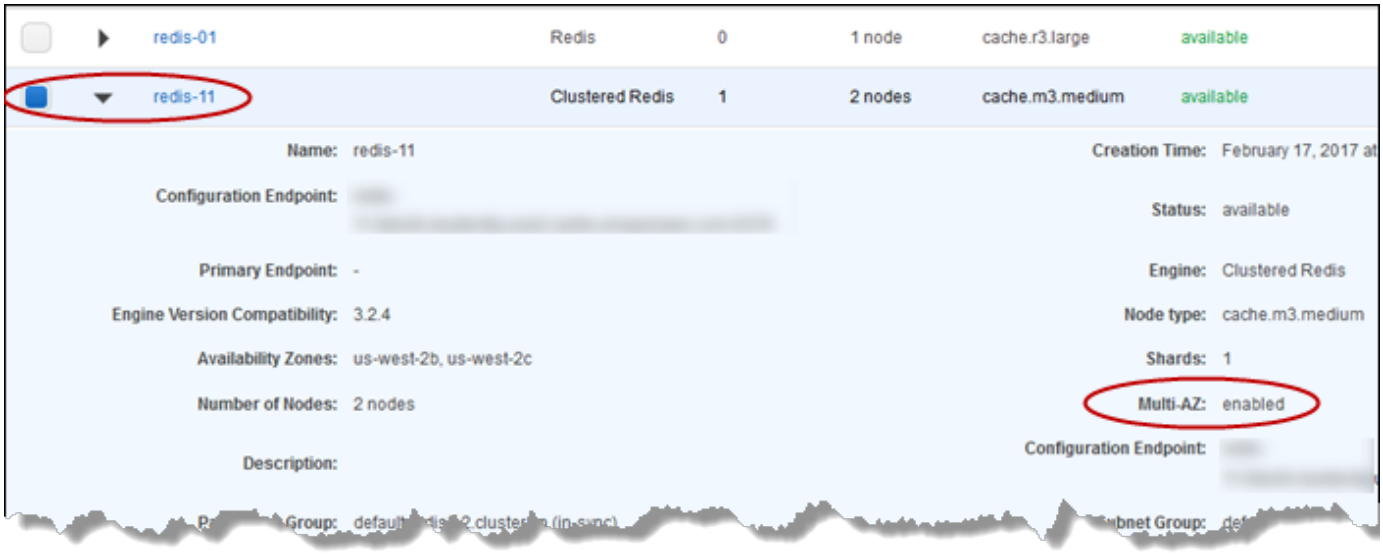

- Testing automatic failover using the AWS Management Console
- Testing automatic failover using the AWS CLI
- Testing automatic failover using the ElastiCache API


### Testing automatic failover using the AWS Management Console

Use the following procedure to test automatic failover with the console.

**To test automatic failover**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**.
3. From the list of clusters, choose the box to the left of the cluster you want to test. This cluster must have at least one read replica node.
4. In the **Details** area, confirm that this cluster is Multi-AZ enabled. If the cluster isn't Multi-AZ enabled, either choose a different cluster or modify this cluster to enable Multi-AZ. For more information, see Using the ElastiCache AWS Management Console.




5. For Valkey or Redis OSS (cluster mode disabled), choose the cluster's name.

   For Valkey or Redis OSS (cluster mode enabled), do the following:

   a. Choose the cluster's name.

b. On the **Shards** page, for the shard (called node group in the API and CLI) on which you want to test failover, choose the shard's name.

6. On the Nodes page, choose **Failover Primary**.
7. Choose **Continue** to fail over the primary, or **Cancel** to cancel the operation and not fail over the primary node.

   During the failover process, the console continues to show the node's status as *available*. To track the progress of your failover test, choose **Events** from the console navigation pane. On the **Events** tab, watch for events that indicate your failover has started (`Test Failover API called`) and completed (`Recovery completed`).

### Testing automatic failover using the AWS CLI

You can test automatic failover on any Multi-AZ enabled cluster using the AWS CLI operation `test-failover`.

**Parameters**

- `--replication-group-id` – Required. The replication group (on the console, cluster) that is to be tested.
- `--node-group-id` – Required. The name of the node group you want to test automatic failover on. You can test a maximum of 15 node groups in a rolling 24-hour period.

The following example uses the AWS CLI to test automatic failover on the node group `redis00-0003` in the Valkey or Redis OSS (cluster mode enabled) cluster `redis00`.

**Example Test automatic failover**

For Linux, macOS, or Unix:

```
aws elasticache test-failover \
   --replication-group-id redis00 \
   --node-group-id 0003
```

For Windows:

```
aws elasticache test-failover ^
```

```
    --replication-group-id redis00 ^
    --node-group-id 0003
```

Output from the preceding command looks something like the following.

```
{
    "ReplicationGroup": {
        "Status": "available",
        "Description": "1 shard, 3 nodes (1 + 2 replicas)",
        "NodeGroups": [
            {
                "Status": "available",
                "NodeGroupMembers": [
                    {
                        "CurrentRole": "primary",
                        "PreferredAvailabilityZone": "us-west-2c",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address":
 "redis1x3-001.7ekv3t.0001.usw2.cache.amazonaws.com"
                        },
                        "CacheClusterId": "redis1x3-001"
                    },
                    {
                        "CurrentRole": "replica",
                        "PreferredAvailabilityZone": "us-west-2a",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address":
 "redis1x3-002.7ekv3t.0001.usw2.cache.amazonaws.com"
                        },
                        "CacheClusterId": "redis1x3-002"
                    },
                    {
                        "CurrentRole": "replica",
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address":
 "redis1x3-003.7ekv3t.0001.usw2.cache.amazonaws.com"
```

```
                        },
                        "CacheClusterId": "redis1x3-003"
                    }
                ],
                "NodeGroupId": "0001",
                "PrimaryEndpoint": {
                    "Port": 6379,
                    "Address": "redis1x3.7ekv3t.ng.0001.usw2.cache.amazonaws.com"
                }
            }
        ],
        "ClusterEnabled": false,
        "ReplicationGroupId": "redis1x3",
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "enabled",
        "MultiAZ": "enabled",
        "SnapshotWindow": "11:30-12:30",
        "SnapshottingClusterId": "redis1x3-002",
        "MemberClusters": [
            "redis1x3-001",
            "redis1x3-002",
            "redis1x3-003"
        ],
        "CacheNodeType": "cache.m5.large",
        "DataTiering": "disabled",
        "PendingModifiedValues": {}
    }
}
```

To track the progress of your failover, use the AWS CLI `describe-events` operation.

For more information, see the following:

- test-failover in the *AWS CLI Command Reference.*
- describe-events in the *AWS CLI Command Reference.*

### Testing automatic failover using the ElastiCache API

You can test automatic failover on any cluster enabled with Multi-AZ using the ElastiCache API operation `TestFailover`.

**Parameters**

- `ReplicationGroupId` – Required. The replication group (on the console, cluster) to be tested.
- `NodeGroupId` – Required. The name of the node group that you want to test automatic failover on. You can test a maximum of 15 node groups in a rolling 24-hour period.

The following example tests automatic failover on the node group `redis00-0003` in the replication group (on the console, cluster) `redis00`.

**Example Testing automatic failover**

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=TestFailover
    &NodeGroupId=0003
    &ReplicationGroupId=redis00
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20140401T192317Z
    &X-Amz-Credential=<credential>
```

To track the progress of your failover, use the ElastiCache `DescribeEvents` API operation.

For more information, see the following:

- TestFailover in the *ElastiCache API Reference*
- DescribeEvents in the *ElastiCache API Reference*

**Limitations on Multi-AZ**

Be aware of the following limitations for Multi-AZ:

- Multi-AZ is supported on all Valkey and Redis OSS versions.
- Valkey and Redis OSS replication is asynchronous. Therefore, when a primary node fails over to a replica, a small amount of data might be lost due to replication lag. For clusters with durability enabled, this limitation does not apply — all committed data is restored from the Multi-AZ transactional log.

When choosing the replica to promote to primary, ElastiCache chooses the replica with the least replication lag. In other words, it chooses the replica that is most current. Doing so helps minimize the amount of lost data. The replica with the least replication lag can be in the same or different Availability Zone from the failed primary node.

- When you manually promote read replicas to primary on Valkey or Redis OSS clusters with cluster mode disabled, you can do so only when Multi-AZ and automatic failover are disabled. To promote a read replica to primary, take the following steps:

  > **Note**
  >
  > Manual replica promotion is different from testing automatic failover. With manual promotion (described here), you must first disable Multi-AZ and automatic failover, and you can choose which replica becomes primary. To test automatic failover, use the `TestFailover` API. This requires automatic failover to be *enabled* and simulates a primary node failure to verify your configuration. For more information, see Testing automatic failover.

  1. Disable Multi-AZ on the cluster.
  2. Disable automatic failover on the cluster. You can do this through the console by clearing the **Auto failover** check box for the replication group. You can also do this using the AWS CLI by setting the `AutomaticFailoverEnabled` property to `false` when calling the `ModifyReplicationGroup` operation.
  3. Promote the read replica to primary.
  4. Re-enable Multi-AZ.

- ElastiCache for Redis OSS Multi-AZ and append-only file (AOF) are mutually exclusive. If you enable one, you can't enable the other.
- A node's failure can be caused by the rare event of an entire Availability Zone failing. In this case, the replica replacing the failed primary is created only when the Availability Zone is back up. For example, consider a replication group with the primary in AZ-a and replicas in AZ-b and AZ-c. If the primary fails, the replica with the least replication lag is promoted to primary cluster. Then, ElastiCache creates a new replica in AZ-a (where the failed primary was located) only when AZ-a is back up and available.
- A customer-initiated reboot of a primary doesn't trigger automatic failover. Other reboots and failures do trigger automatic failover.

- When the primary is rebooted, it's cleared of data when it comes back online. When the read replicas see the cleared primary cluster, they clear their copy of the data, which causes data loss.
- After a read replica has been promoted, the other replicas sync with the new primary. After the initial sync, the replicas' content is deleted and they sync the data from the new primary. This sync process causes a brief interruption, during which the replicas are not accessible. The sync process also causes a temporary load increase on the primary while syncing with the replicas. This behavior is native to Valkey and Redis OSS and isn't unique to ElastiCache Multi-AZ. For details about this behavior, see Replication on the Valkey website. For clusters with durability enabled, replicas restore from the transactional log independently without imposing load on the primary.

> **Important**
>
> For Valkey and Redis OSS, you can't create external replicas.

## Synchronization and backup with durability

For clusters with durability enabled, synchronization and backup operations differ from standard clusters:

- **Off-box snapshotting:** Snapshots are created by ephemeral instances that read from the Multi-AZ transactional log, eliminating performance impact on your cluster.
- **Log-based recovery:** Failed replicas restore from the transactional log and snapshots rather than requiring a full synchronization from the primary.

## Creating a Valkey or Redis OSS replication group

You have the following options for creating a cluster with replica nodes. One applies when you already have an available Valkey or Redis OSS (cluster mode disabled) cluster not associated with any cluster that has replicas to use as the primary node. The other applies when you need to create a primary node with the cluster and read replicas. Currently, a Valkey or Redis OSS (cluster mode enabled) cluster must be created from scratch.

**Option 1: Creating a replication group using an existing cluster**

Use this option to leverage an existing single-node Valkey or Redis OSS (cluster mode disabled) cluster. You specify this existing node as the primary node in the new cluster, and then individually add 1 to 5 read replicas to the cluster. If the existing cluster is active, read replicas synchronize with it as they are created. See Creating a replication group using an existing cluster.

> **Important**
>
> You cannot create a Valkey or Redis OSS (cluster mode enabled) cluster using an existing cluster. To create a Valkey or Redis OSS (cluster mode enabled) cluster (API/CLI: replication group) using the ElastiCache console, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console).

**Option 2: Creating a Valkey or Redis OSS replication group from scratch**

Use this option if you don't already have an available Valkey or Redis OSS (cluster mode disabled) cluster to use as the cluster's primary node, or if you want to create a Valkey or Redis OSS (cluster mode enabled) cluster. See Creating a Valkey or Redis OSS replication group from scratch.

### Creating a replication group using an existing cluster

The following procedure adds a replication group to your Valkey or Redis OSS (cluster mode disabled) single-node cluster, which is necessary in order to upgrade your cluster to the latest version of Valkey. This is an in-place procedure that involves zero downtime and zero data loss. When you create a replication group for your single-node cluster, the cluster's node becomes the primary node in the new cluster. If you do not have a Valkey or Redis OSS (cluster mode disabled) cluster that you can use as the new cluster's primary, see Creating a Valkey or Redis OSS replication group from scratch.

An available cluster is an existing single-node Valkey or Redis OSS cluster. Currently, Valkey or Redis OSS (cluster mode enabled) does not support creating a cluster with replicas using an available single-node cluster. If you want to create a Valkey or Redis OSS (cluster mode enabled) cluster, see Creating a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console).

#### Creating a replication group using an existing cluster (Console)

See the topic Using the ElastiCache AWS Management Console.

#### Creating a replication group using an available Valkey or Redis OSS cluster (AWS CLI)

There are two steps to creating a replication group with read replicas when using an available Valkey or Redis OSS Cache Cluster for the primary when using the AWS CLI.

When using the AWS CLI you create a replication group specifying the available standalone node as the cluster's primary node, `--primary-cluster-id` and the number of nodes you want in the cluster using the CLI command, `create-replication-group`. Include the following parameters.

**--replication-group-id**

The name of the replication group you are creating. The value of this parameter is used as the basis for the names of the added nodes with a sequential 3-digit number added to the end of the `--replication-group-id`. For example, `sample-repl-group-001`.

Valkey or Redis OSS (cluster mode disabled) replication group naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.

- Can't end with a hyphen.

**--replication-group-description**

Description of the replication group.

**--num-node-groups**

The number of nodes you want in this cluster. This value includes the primary node. This parameter has a maximum value of six.

**--primary-cluster-id**

The name of the available Valkey or Redis OSS (cluster mode disabled) cluster's node that you want to be the primary node in this replication group.

The following command creates the replication group `sample-repl-group` using the available Valkey or Redis OSS (cluster mode disabled) cluster `redis01` as the replication group's primary node. It creates 2 new nodes which are read replicas. The settings of `redis01` (that is, parameter group, security group, node type, engine version, and so on.) will be applied to all nodes in the replication group.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
   --replication-group-id sample-repl-group \
   --replication-group-description "demo cluster with replicas" \
   --num-cache-clusters 3 \
   --primary-cluster-id redis01
```

For Windows:

```
aws elasticache create-replication-group ^
   --replication-group-id sample-repl-group ^
   --replication-group-description "demo cluster with replicas" ^
   --num-cache-clusters 3 ^
   --primary-cluster-id redis01
```

For additional information and parameters you might want to use, see the AWS CLI topic create-replication-group.

**Next, add read replicas to the replication group**

After the replication group is created, add one to five read replicas to it using the `create-cache-cluster` command, being sure to include the following parameters.

**--cache-cluster-id**

The name of the cluster you are adding to the replication group.

Cluster naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**--replication-group-id**

The name of the replication group to which you are adding this cluster.

Repeat this command for each read replica you want to add to the replication group, changing only the value of the `--cache-cluster-id` parameter.

> **Note**
>
> Remember, a replication group cannot have more than five read replicas. Attempting to add a read replica to a replication group that already has five read replicas causes the operation to fail.

The following code adds the read replica `my-replica01` to the replication group `sample-repl-group`. The settings of the primary cluster–parameter group, security group, node type, and so on.–will be applied to nodes as they are added to the replication group.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
   --cache-cluster-id my-replica01 \
   --replication-group-id sample-repl-group
```

For Windows:

```
aws elasticache create-cache-cluster ^
   --cache-cluster-id my-replica01 ^
   --replication-group-id sample-repl-group
```

Output from this command will look something like this.

```
{
    "ReplicationGroup": {
        "Status": "creating",
        "Description": "demo cluster with replicas",
        "ClusterEnabled": false,
        "ReplicationGroupId": "sample-repl-group",
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "disabled",
        "SnapshotWindow": "00:00-01:00",
        "SnapshottingClusterId": "redis01",
        "MemberClusters": [
            "sample-repl-group-001",
            "sample-repl-group-002",
            "redis01"
        ],
        "CacheNodeType": "cache.m4.large",
        "DataTiering": "disabled",
        "PendingModifiedValues": {}
    }
}
```

For additional information, see the AWS CLI topics:

- create-replication-group
- modify-replication-group

### Adding replicas to a standalone Valkey or Redis OSS (Cluster Mode Disabled) cluster (ElastiCache API)

When using the ElastiCache API, you create a replication group specifying the available standalone node as the cluster's primary node, `PrimaryClusterId` and the number of nodes you want in the cluster using the CLI command, `CreateReplicationGroup`. Include the following parameters.

**ReplicationGroupId**

The name of the replication group you are creating. The value of this parameter is used as the basis for the names of the added nodes with a sequential 3-digit number added to the end of the `ReplicationGroupId`. For example, `sample-repl-group-001`.

Valkey or Redis OSS (cluster mode disabled) replication group naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**ReplicationGroupDescription**

Description of the cluster with replicas.

**NumCacheClusters**

The number of nodes you want in this cluster. This value includes the primary node. This parameter has a maximum value of six.

**PrimaryClusterId**

The name of the available Valkey or Redis OSS (cluster mode disabled) cluster that you want to be the primary node in this cluster.

The following command creates the cluster with replicas `sample-repl-group` using the available Valkey or Redis OSS (cluster mode disabled) cluster `redis01` as the replication group's primary node. It creates 2 new nodes which are read replicas. The settings of `redis01` (that is, parameter group, security group, node type, engine version, and so on.) will be applied to all nodes in the replication group.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=CreateReplicationGroup
   &Engine=redis
   &EngineVersion=6.0
   &ReplicationGroupDescription=Demo%20cluster%20with%20replicas
   &ReplicationGroupId=sample-repl-group
   &PrimaryClusterId=redis01
   &Version=2015-02-02
```

```
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

For additional information, see the ElastiCache APL topics:

- CreateReplicationGroup
- ModifyReplicationGroup

**Next, add read replicas to the replication group**

After the replication group is created, add one to five read replicas to it using the `CreateCacheCluster` operation, being sure to include the following parameters.

**CacheClusterId**

The name of the cluster you are adding to the replication group.

Cluster naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**ReplicationGroupId**

The name of the replication group to which you are adding this cluster.

Repeat this operation for each read replica you want to add to the replication group, changing only the value of the `CacheClusterId` parameter.

The following code adds the read replica `myReplica01` to the replication group `myReplGroup` The settings of the primary cluster–parameter group, security group, node type, and so on.–will be applied to nodes as they are added to the replication group.

```
https://elasticache.us-west-2.amazonaws.com/
 ?Action=CreateCacheCluster
 &CacheClusterId=myReplica01
```

```
&ReplicationGroupId=myReplGroup
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2015-02-02
&X-Amz-Algorithm=&AWS;4-HMAC-SHA256
&X-Amz-Credential=[your-access-key-id]/20150202/us-west-2/elasticache/aws4_request
&X-Amz-Date=20150202T170651Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=[signature-value]
```

For additional information and parameters you might want to use, see the ElastiCache API topic CreateCacheCluster.

### Creating a Valkey or Redis OSS replication group from scratch

Following, you can find how to create a Valkey or Redis OSS replication group without using an existing Valkey or Redis OSS cluster as the primary. You can create a Valkey or Redis OSS (cluster mode disabled) or Valkey or Redis OSS (cluster mode enabled) replication group from scratch using the ElastiCache console, the AWS CLI, or the ElastiCache API.

Before you continue, decide whether you want to create a Valkey or Redis OSS (cluster mode disabled) or a Valkey or Redis OSS (cluster mode enabled) replication group. For guidance in deciding, see Replication: Valkey and Redis OSS Cluster Mode Disabled vs. Enabled.

#### Topics

- Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch
- Creating a replication group in Valkey or Redis OSS (Cluster Mode Enabled) from scratch

### Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch

You can create a Valkey or Redis OSS (cluster mode disabled) replication group from scratch using the ElastiCache console, the AWS CLI, or the ElastiCache API. A Valkey or Redis OSS (cluster mode disabled) replication group always has one node group, a primary cluster, and up to five read replicas. Valkey or Redis OSS (cluster mode disabled) replication groups don't support partitioning your data.

> **Note**
>
> The node/shard limit can be increased to a maximum of 500 per cluster. To request a limit increase, see AWS Service Limits and include the instance type in the request.

To create a Valkey or Redis OSS (cluster mode disabled) replication group from scratch, take one of the following approaches:

### Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch (AWS CLI)

The following procedure creates a Valkey or Redis OSS (cluster mode disabled) replication group using the AWS CLI.

When you create a Valkey or Redis OSS (cluster mode disabled) replication group from scratch, you create the replication group and all its nodes with a single call to the AWS CLI `create-replication-group` command. Include the following parameters.

**--replication-group-id**

The name of the replication group you are creating.

Valkey or Redis OSS (cluster mode disabled) replication group naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**--replication-group-description**

Description of the replication group.

**--num-cache-clusters**

The number of nodes you want created with this replication group, primary and read replicas combined.

If you enable Multi-AZ (`--automatic-failover-enabled`), the value of `--num-cache-clusters` must be at least 2.

**--cache-node-type**

The node type for each node in the replication group.

ElastiCache supports the following node types. Generally speaking, the latest generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For performance details for each node type, see Amazon EC2 Instance Types.

**--data-tiering-enabled**

Set this parameter if you are using a data tiering node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see Data tiering in ElastiCache.

**--cache-parameter-group**

Specify a parameter group that corresponds to your engine version. For example, if you are running Redis OSS 7.1, specify the `default.redis7` parameter group or a parameter group derived from `default.redis7` to create a Valkey or Redis OSS (cluster mode disabled) replication group. For more information, see Valkey and Redis OSS parameters.

**--network-type**

Either `ipv4`, `ipv6` or `dual-stack`. If you choose dual-stack, you must set the `--IpDiscovery` parameter to either `ipv4` or `ipv6`.

**--engine**

redis

**--engine-version**

To have the richest set of features, choose the latest engine version.

The names of the nodes will be derived from the replication group name by postpending `-00#` to the replication group name. For example, using the replication group name `myReplGroup`,

the name for the primary will be `myReplGroup-001` and the read replicas `myReplGroup-002` through `myReplGroup-006`.

If you want to enable in-transit or at-rest encryption on this replication group, add either or both of the `--transit-encryption-enabled` or `--at-rest-encryption-enabled` parameters and meet the following conditions.

- Your replication group must be running Valkey, or Redis OSS version 4.0.10 or later.
- The replication group must be created in an Amazon VPC.
- You must also include the parameter `--cache-subnet-group`.
- You must also include the parameter `--auth-token` with the customer specified string value for your AUTH token (password) needed to perform operations on this replication group.

The following operation creates a Valkey or Redis OSS (cluster mode disabled) replication group `sample-repl-group` with three nodes, a primary and two replicas.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
   --replication-group-id sample-repl-group \
   --replication-group-description "Demo cluster with replicas" \
   --num-cache-clusters 3 \
   --cache-node-type cache.m4.large \
   --engine redis
```

For Windows:

```
aws elasticache create-replication-group ^
   --replication-group-id sample-repl-group ^
   --replication-group-description "Demo cluster with replicas" ^
   --num-cache-clusters 3 ^
   --cache-node-type cache.m4.large ^
   --engine redis
```

Output from the this command is something like this.

```
{
    "ReplicationGroup": {
        "Status": "creating",
        "Description": "Demo cluster with replicas",
```

```
        "ClusterEnabled": false,
        "ReplicationGroupId": "sample-repl-group",
        "SnapshotRetentionLimit": 0,
        "AutomaticFailover": "disabled",
        "SnapshotWindow": "01:30-02:30",
        "MemberClusters": [
            "sample-repl-group-001",
            "sample-repl-group-002",
            "sample-repl-group-003"
        ],
        "CacheNodeType": "cache.m4.large",
        "DataTiering": "disabled",
        "PendingModifiedValues": {}
    }
}
```

For additional information and parameters you might want to use, see the AWS CLI topic create-replication-group.

**Creating a Valkey or Redis OSS (cluster mode disabled) replication group from scratch (ElastiCache API)**

The following procedure creates a Valkey or Redis OSS (cluster mode disabled) replication group using the ElastiCache API.

When you create a Valkey or Redis OSS (cluster mode disabled) replication group from scratch, you create the replication group and all its nodes with a single call to the ElastiCache API `CreateReplicationGroup` operation. Include the following parameters.

**ReplicationGroupId**

The name of the replication group you are creating.

Valkey or Redis OSS (cluster mode enabled) replication group naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**ReplicationGroupDescription**

Your description of the replication group.

**NumCacheClusters**

The total number of nodes you want created with this replication group, primary and read replicas combined.

If you enable Multi-AZ (`AutomaticFailoverEnabled=true`), the value of `NumCacheClusters` must be at least 2.

**CacheNodeType**

The node type for each node in the replication group.

ElastiCache supports the following node types. Generally speaking, the latest generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For performance details for each node type, see Amazon EC2 Instance Types.

**--data-tiering-enabled**

Set this parameter if you are using a data tiering node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see Data tiering in ElastiCache.

**CacheParameterGroup**

Specify a parameter group that corresponds to your engine version. For example, if you are running Redis OSS 7.1, specify the `default.redis7` parameter group or a parameter group derived from `default.redis7` to create a Valkey or Redis OSS (cluster mode disabled) replication group. For more information, see Valkey and Redis OSS parameters.

**--network-type**

Either `ipv4`, `ipv` or `dual-stack`. If you choose dual-stack, you must set the `--IpDiscovery` parameter to either `ipv4` or `ipv6`.

**Engine**

redis

**EngineVersion**

6.0

The names of the nodes will be derived from the replication group name by postpending `-00#` to the replication group name. For example, using the replication group name `myReplGroup`,

the name for the primary will be `myReplGroup-001` and the read replicas `myReplGroup-002` through `myReplGroup-006`.

If you want to enable in-transit or at-rest encryption on this replication group, add either or both of the `TransitEncryptionEnabled=true` or `AtRestEncryptionEnabled=true` parameters and meet the following conditions.

- Your replication group must be running Valkey, or Redis OSS version 4.0.10 or later.
- The replication group must be created in an Amazon VPC.
- You must also include the parameter `CacheSubnetGroup`.
- You must also include the parameter `AuthToken` with the customer specified string value for your AUTH token (password) needed to perform operations on this replication group.

The following operation creates the Valkey or Redis OSS (cluster mode disabled) replication group `myReplGroup` with three nodes, a primary and two replicas.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=CreateReplicationGroup
   &CacheNodeType=cache.m4.large
   &CacheParameterGroup=default.redis6.x
   &Engine=redis
   &EngineVersion=6.0
   &NumCacheClusters=3
   &ReplicationGroupDescription=test%20group
   &ReplicationGroupId=myReplGroup
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

For additional information and parameters you might want to use, see the ElastiCache API topic CreateReplicationGroup.

### Creating a replication group in Valkey or Redis OSS (Cluster Mode Enabled) from scratch

You can create a Valkey or Redis OSS (cluster mode enabled) cluster (API/CLI: *replication group*) using the ElastiCache console, the AWS CLI, or the ElastiCache API. A Valkey or Redis OSS (cluster mode enabled) replication group has from 1 to 500 shards (API/CLI: node groups), a primary node in each shard, and up to 5 read replicas in each shard. You can create a cluster with higher number of shards and lower number of replicas totaling up to 90 nodes per cluster. This cluster configuration can range from 90 shards and 0 replicas to 15 shards and 5 replicas, which is the maximum number of replicas allowed.

The node or shard limit can be increased to a maximum of 500 per cluster if the Valkey or Redis OSS engine version is 5.0.6 or higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). Make sure there are enough available IP addresses to accommodate the increase. Common pitfalls include the subnets in the subnet group have too small a CIDR range or the subnets are shared and heavily used by other clusters. For more information, see Creating a subnet group.

For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

#### Creating a Cluster in Valkey or Redis OSS (Cluster Mode Enabled)

- Creating a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console)
- Creating a Valkey or Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)
- Creating a replication group in Valkey or Redis OSS (Cluster Mode Enabled) from scratch (ElastiCache API)

#### Creating a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console)

To create a Valkey or Redis OSS (cluster mode enabled) cluster, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console). Be sure to enable cluster mode, **Cluster Mode enabled (Scale Out)**, and specify at least two shards and one replica node in each.

### Creating a Valkey or Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)

The following procedure creates a Valkey or Redis OSS (cluster mode enabled) replication group using the AWS CLI.

When you create a Valkey or Redis OSS (cluster mode enabled) replication group from scratch, you create the replication group and all its nodes with a single call to the AWS CLI `create-replication-group` command. Include the following parameters.

**--replication-group-id**

The name of the replication group you are creating.

Valkey or Redis OSS (cluster mode enabled) replication group naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**--replication-group-description**

Description of the replication group.

**--cache-node-type**

The node type for each node in the replication group.

ElastiCache supports the following node types. Generally speaking, the latest generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For performance details for each node type, see Amazon EC2 Instance Types.

**--data-tiering-enabled**

Set this parameter if you are using a data tiering node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see Data tiering in ElastiCache.

**--cache-parameter-group**

Specify the `default.redis6.x.cluster.on` parameter group or a parameter group derived from `default.redis6.x.cluster.on` to create a Valkey or Redis OSS (cluster mode enabled) replication group. For more information, see Redis OSS 6.x parameter changes.

**--engine**

redis

**--engine-version**

7.1

**--num-node-groups**

The number of node groups in this replication group. Valid values are 1 to 500.

> **Note**
>
> The node/shard limit can be increased to a maximum of 500 per cluster. To request a limit increase, see AWS Service Limits and select limit type "Nodes per cluster per instance type".

**--replicas-per-node-group**

The number of replica nodes in each node group. Valid values are 0 to 5.

**--network-type**

Either `ipv4`, `ipv` or `dual-stack`. If you choose dual-stack, you must set the `--IpDiscovery` parameter to either `ipv4` or `ipv6`.

If you want to enable in-transit or at-rest encryption on this replication group, add either or both of the `--transit-encryption-enabled` or `--at-rest-encryption-enabled` parameters and meet the following conditions.

- Your replication group must be running Valkey, or Redis OSS version 4.0.10 or later.
- The replication group must be created in an Amazon VPC.
- You must also include the parameter `--cache-subnet-group`.
- You must also include the parameter `--auth-token` with the customer specified string value for your AUTH token (password) needed to perform operations on this replication group.

The following operation creates the Valkey or Redis OSS (cluster mode enabled) replication group `sample-repl-group` with three node groups/shards (--num-node-groups), each with three nodes, a primary and two read replicas (--replicas-per-node-group).

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
   --replication-group-id sample-repl-group \
   --replication-group-description "Demo cluster with replicas" \
   --num-node-groups 3 \
   --replicas-per-node-group 2 \
   --cache-node-type cache.m4.large \
   --engine redis \
   --security-group-ids SECURITY_GROUP_ID \
   --cache-subnet-group-name SUBNET_GROUP_NAME>
```

For Windows:

```
aws elasticache create-replication-group ^
   --replication-group-id sample-repl-group ^
   --replication-group-description "Demo cluster with replicas" ^
   --num-node-groups 3 ^
   --replicas-per-node-group 2 ^
   --cache-node-type cache.m4.large ^
   --engine redis ^
   --security-group-ids SECURITY_GROUP_ID ^
   --cache-subnet-group-name SUBNET_GROUP_NAME>
```

The preceding command generates the following output.

```
{
    "ReplicationGroup": {
        "Status": "creating",
        "Description": "Demo cluster with replicas",
        "ReplicationGroupId": "sample-repl-group",
        "SnapshotRetentionLimit": 0,
        "AutomaticFailover": "enabled",
        "SnapshotWindow": "05:30-06:30",
        "MemberClusters": [
            "sample-repl-group-0001-001",
            "sample-repl-group-0001-002",
            "sample-repl-group-0001-003",
            "sample-repl-group-0002-001",
            "sample-repl-group-0002-002",
```

```
            "sample-repl-group-0002-003",
            "sample-repl-group-0003-001",
            "sample-repl-group-0003-002",
            "sample-repl-group-0003-003"
        ],
        "PendingModifiedValues": {}
    }
}
```

When you create a Valkey or Redis OSS (cluster mode enabled) replication group from scratch, you are able to configure each shard in the cluster using the `--node-group-configuration` parameter as shown in the following example which configures two node groups (Console: shards). The first shard has two nodes, a primary and one read replica. The second shard has three nodes, a primary and two read replicas.

**--node-group-configuration**

The configuration for each node group. The `--node-group-configuration` parameter consists of the following fields.

- `PrimaryAvailabilityZone` – The Availability Zone where the primary node of this node group is located. If this parameter is omitted, ElastiCache chooses the Availability Zone for the primary node.

  **Example:** us-west-2a.

- `ReplicaAvailabilityZones` – A comma separated list of Availability Zones where the read replicas are located. The number of Availability Zones in this list must match the value of `ReplicaCount`. If this parameter is omitted, ElastiCache chooses the Availability Zones for the replica nodes.

  **Example:** "us-west-2a,us-west-2b,us-west-2c"

- `ReplicaCount` – The number of replica nodes in this node group.
- `Slots` – A string that specifies the keyspace for the node group. The string is in the format `startKey-endKey`. If this parameter is omitted, ElastiCache allocates keys equally among the node groups.

  **Example:** "0-4999"

The following operation creates the Valkey or Redis OSS (cluster mode enabled) replication group `new-group` with two node groups/shards (`--num-node-groups`). Unlike the preceding example, each node group is configured differently from the other node group (`--node-group-configuration`).

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
  --replication-group-id new-group \
  --replication-group-description "Sharded replication group" \
  --engine redis \
  --snapshot-retention-limit 8 \
  --cache-node-type cache.m4.medium \
  --num-node-groups 2 \
  --node-group-configuration \
      "ReplicaCount=1,Slots=0-8999,PrimaryAvailabilityZone='us-
east-1c',ReplicaAvailabilityZones='us-east-1b'" \
      "ReplicaCount=2,Slots=9000-16383,PrimaryAvailabilityZone='us-
east-1a',ReplicaAvailabilityZones='us-east-1a','us-east-1c'"
```

For Windows:

```
aws elasticache create-replication-group ^
  --replication-group-id new-group ^
  --replication-group-description "Sharded replication group" ^
  --engine redis ^
  --snapshot-retention-limit 8 ^
  --cache-node-type cache.m4.medium ^
  --num-node-groups 2 ^
  --node-group-configuration \
      "ReplicaCount=1,Slots=0-8999,PrimaryAvailabilityZone='us-
east-1c',ReplicaAvailabilityZones='us-east-1b'" \
      "ReplicaCount=2,Slots=9000-16383,PrimaryAvailabilityZone='us-
east-1a',ReplicaAvailabilityZones='us-east-1a','us-east-1c'"
```

The preceding operation generates the following output.

```
{
    "ReplicationGroup": {
        "Status": "creating",
        "Description": "Sharded replication group",
        "ReplicationGroupId": "rc-rg",
```

```
        "SnapshotRetentionLimit": 8,
        "AutomaticFailover": "enabled",
        "SnapshotWindow": "10:00-11:00",
        "MemberClusters": [
            "rc-rg-0001-001",
            "rc-rg-0001-002",
            "rc-rg-0002-001",
            "rc-rg-0002-002",
            "rc-rg-0002-003"
        ],
        "PendingModifiedValues": {}
    }
}
```

For additional information and parameters you might want to use, see the AWS CLI topic create-replication-group.

**Creating a replication group in Valkey or Redis OSS (Cluster Mode Enabled) from scratch (ElastiCache API)**

The following procedure creates a Valkey or Redis OSS (cluster mode enabled) replication group using the ElastiCache API.

When you create a Valkey or Redis OSS (cluster mode enabled) replication group from scratch, you create the replication group and all its nodes with a single call to the ElastiCache API `CreateReplicationGroup` operation. Include the following parameters.

**ReplicationGroupId**

The name of the replication group you are creating.

Valkey or Redis OSS (cluster mode enabled) replication group naming constraints are as follows:

- Must contain 1–40 alphanumeric characters or hyphens.
- Must begin with a letter.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

**ReplicationGroupDescription**

Description of the replication group.

**NumNodeGroups**

The number of node groups you want created with this replication group. Valid values are 1 to 500.

**ReplicasPerNodeGroup**

The number of replica nodes in each node group. Valid values are 1 to 5.

**NodeGroupConfiguration**

The configuration for each node group. The `NodeGroupConfiguration` parameter consists of the following fields.

- `PrimaryAvailabilityZone` – The Availability Zone where the primary node of this node group is located. If this parameter is omitted, ElastiCache chooses the Availability Zone for the primary node.

  **Example:** us-west-2a.

- `ReplicaAvailabilityZones` – A list of Availability Zones where the read replicas are located. The number of Availability Zones in this list must match the value of `ReplicaCount`. If this parameter is omitted, ElastiCache chooses the Availability Zones for the replica nodes.
- `ReplicaCount` – The number of replica nodes in this node group.
- `Slots` – A string that specifies the keyspace for the node group. The string is in the format `startKey-endKey`. If this parameter is omitted, ElastiCache allocates keys equally among the node groups.

  **Example:** "0-4999"

**CacheNodeType**

The node type for each node in the replication group.

ElastiCache supports the following node types. Generally speaking, the latest generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For performance details for each node type, see Amazon EC2 Instance Types.

**--data-tiering-enabled**

Set this parameter if you are using a data tiering node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see Data tiering in ElastiCache.

**CacheParameterGroup**

Specify the `default.redis6.x.cluster.on` parameter group or a parameter group derived from `default.redis6.x.cluster.on` to create a Valkey or Redis OSS (cluster mode enabled) replication group. For more information, see Redis OSS 6.x parameter changes.

**--network-type**

Either `ipv4`, `ipv` or `dual-stack`. If you choose dual-stack, you must set the `--IpDiscovery` parameter to either `ipv4` or `ipv6`.

**Engine**

redis

**EngineVersion**

6.0

If you want to enable in-transit or at-rest encryption on this replication group, add either or both of the `TransitEncryptionEnabled=true` or `AtRestEncryptionEnabled=true` parameters and meet the following conditions.

- Your replication group must be running Valkey, or Redis OSS version 4.0.10 or later.
- The replication group must be created in an Amazon VPC.
- You must also include the parameter `CacheSubnetGroup`.
- You must also include the parameter `AuthToken` with the customer specified string value for your AUTH token (password) needed to perform operations on this replication group.

Line breaks are added for ease of reading.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=CreateReplicationGroup
   &CacheNodeType=cache.m4.large
   &CacheParemeterGroup=default.redis6.xcluster.on
   &Engine=redis
   &EngineVersion=6.0
```

```
    &NumNodeGroups=3
    &ReplicasPerNodeGroup=2
    &ReplicationGroupDescription=test%20group
    &ReplicationGroupId=myReplGroup
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For additional information and parameters you might want to use, see the ElastiCache API topic CreateReplicationGroup.

## Viewing a replication group's details

There are times you may want to view the details of a replication group. You can use the ElastiCache console, the AWS CLI for ElastiCache, or the ElastiCache API. The console process is different for Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled).

### Viewing a Replication Group's Details

- Viewing a Valkey or Redis OSS (Cluster Mode Disabled) with replicas
  - Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Replication Group (Console)
  - Viewing a Valkey or Redis OSS (Cluster Mode Disabled) replication group (AWS CLI)
  - Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Replication Group (ElastiCache API)
- Viewing a replication group: Valkey or Redis OSS (Cluster Mode Enabled)
  - Viewing a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console)
  - Viewing a Valkey or Redis OSS (Cluster Mode Enabled) cluster (AWS CLI)
  - Viewing a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (ElastiCache API)
- Viewing a replication group's details (AWS CLI)
- Viewing a replication group's details (ElastiCache API)

### Viewing a Valkey or Redis OSS (Cluster Mode Disabled) with replicas

You can view the details of a Valkey or Redis OSS (cluster mode disabled) cluster with replicas (API/CLI: *replication group*) using the ElastiCache console, the AWS CLI for ElastiCache, or the ElastiCache API.

### Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Details

- Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Replication Group (Console)
- Viewing a Valkey or Redis OSS (Cluster Mode Disabled) replication group (AWS CLI)
- Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Replication Group (ElastiCache API)

#### Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Replication Group (Console)

To view the details of a Valkey or Redis OSS (cluster mode disabled) cluster with replicas using the ElastiCache console, see the topic Viewing Valkey or Redis OSS (Cluster Mode Disabled) details (Console).

#### Viewing a Valkey or Redis OSS (Cluster Mode Disabled) replication group (AWS CLI)

For an AWS CLI example that displays a Valkey or Redis OSS (cluster mode disabled) replication group's details, see Viewing a replication group's details (AWS CLI).

#### Viewing a Valkey or Redis OSS (Cluster Mode Disabled) Replication Group (ElastiCache API)

For an ElastiCache API example that displays a Valkey or Redis OSS (cluster mode disabled) replication group's details, see Viewing a replication group's details (ElastiCache API).

### Viewing a replication group: Valkey or Redis OSS (Cluster Mode Enabled)

#### Viewing a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console)

To view the details of a Valkey or Redis OSS (cluster mode enabled) cluster using the ElastiCache console, see Viewing details for a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console).

#### Viewing a Valkey or Redis OSS (Cluster Mode Enabled) cluster (AWS CLI)

For an ElastiCache CLI example that displays a Valkey or Redis OSS (cluster mode enabled) replication group's details, see Viewing a replication group's details (AWS CLI).

#### Viewing a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (ElastiCache API)

For an ElastiCache API example that displays a Valkey or Redis OSS (cluster mode enabled) replication group's details, see Viewing a replication group's details (ElastiCache API).

### Viewing a replication group's details (AWS CLI)

You can view the details for a replication group using the AWS CLI `describe-replication-groups` command. Use the following optional parameters to refine the listing. Omitting the parameters returns the details for up to 100 replication groups.

#### Optional Parameters

- `--replication-group-id` – Use this parameter to list the details of a specific replication group. If the specified replication group has more than one node group, results are returned grouped by node group.
- `--max-items` – Use this parameter to limit the number of replication groups listed. The value of `--max-items` cannot be less than 20 or greater than 100.

#### Example

The following code lists the details for up to 100 replication groups.

```
aws elasticache describe-replication-groups
```

The following code lists the details for `sample-repl-group`.

```
aws elasticache describe-replication-groups --replication-group-id sample-repl-group
```

The following code lists the details for `sample-repl-group`.

```
aws elasticache describe-replication-groups --replication-group-id sample-repl-group
```

The following code list the details for up to 25 replication groups.

```
aws elasticache describe-replication-groups --max-items 25
```

Output from this operation should look something like this (JSON format).

```
{
   "ReplicationGroups": [
     {
       "Status": "available",
       "Description": "test",
       "NodeGroups": [
```

```
        {
           "Status": "available",
              "NodeGroupMembers": [
                 {
                    "CurrentRole": "primary",
                    "PreferredAvailabilityZone": "us-west-2a",
                    "CacheNodeId": "0001",
                    "ReadEndpoint": {
                       "Port": 6379,
                       "Address": "rg-name-001.1abc4d.0001.usw2.cache.amazonaws.com"
                    },
                    "CacheClusterId": "rg-name-001"
                 },
                 {
                    "CurrentRole": "replica",
                    "PreferredAvailabilityZone": "us-west-2b",
                    "CacheNodeId": "0001",
                    "ReadEndpoint": {
                       "Port": 6379,
                       "Address": "rg-name-002.1abc4d.0001.usw2.cache.amazonaws.com"
                    },
                    "CacheClusterId": "rg-name-002"
                 },
                 {
                    "CurrentRole": "replica",
                    "PreferredAvailabilityZone": "us-west-2c",
                    "CacheNodeId": "0001",
                    "ReadEndpoint": {
                       "Port": 6379,
                       "Address": "rg-name-003.1abc4d.0001.usw2.cache.amazonaws.com"
                    },
                    "CacheClusterId": "rg-name-003"
                 }
              ],
              "NodeGroupId": "0001",
              "PrimaryEndpoint": {
                 "Port": 6379,
                 "Address": "rg-name.1abc4d.ng.0001.usw2.cache.amazonaws.com"
              }
           }
        ],
        "ReplicationGroupId": "rg-name",
        "AutomaticFailover": "enabled",
        "SnapshottingClusterId": "rg-name-002",
```

```
         "MemberClusters": [
            "rg-name-001",
            "rg-name-002",
            "rg-name-003"
         ],
         "PendingModifiedValues": {}
      },
      {
      ... some output omitted for brevity
      }
   ]
}
```

For more information, see the AWS CLI for ElastiCache topic describe-replication-groups.

### Viewing a replication group's details (ElastiCache API)

You can view the details for a replication using the AWS CLI `DescribeReplicationGroups` operation. Use the following optional parameters to refine the listing. Omitting the parameters returns the details for up to 100 replication groups.

#### Optional Parameters

- `ReplicationGroupId` – Use this parameter to list the details of a specific replication group. If the specified replication group has more than one node group, results are returned grouped by node group.
- `MaxRecords` – Use this parameter to limit the number of replication groups listed. The value of `MaxRecords` cannot be less than 20 or greater than 100. The default is 100.

#### Example

The following code list the details for up to 100 replication groups.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeReplicationGroups
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

The following code lists the details for `myReplGroup`.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeReplicationGroups
   &ReplicationGroupId=myReplGroup
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

The following code list the details for up to 25 clusters.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeReplicationGroups
   &MaxRecords=25
   &Version=2015-02-02
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

For more information, see the ElastiCache API reference topic DescribeReplicationGroups.

## Finding replication group endpoints

An application can connect to any node in a replication group, provided that it has the DNS endpoint and port number for that node. Depending upon whether you are running a Valkey or Redis OSS (cluster mode disabled) or a Valkey or Redis OSS (cluster mode enabled) replication group, you will be interested in different endpoints.

### Valkey or Redis OSS (cluster mode disabled)

Valkey or Redis OSS (cluster mode disabled) clusters with replicas have three types of endpoints; the *primary endpoint*, the *reader endpoint* and the *node endpoints*. The primary endpoint is a DNS name that always resolves to the primary node in the cluster. The primary endpoint is immune to changes to your cluster, such as promoting a read replica to the primary role. For write activity, we recommend that your applications connect to the primary endpoint.

A reader endpoint will evenly split incoming connections to the endpoint between all read replicas in an ElastiCache cluster. Additional factors such as when the application creates the connections or how the application (re)-uses the connections will determine the traffic distribution. Reader endpoints keep up with cluster changes in real-time as replicas are added or removed. You can place your ElastiCache for Redis OSS cluster's multiple read replicas in different AWS Availability Zones (AZ) to ensure high availability of reader endpoints.

> **Note**
>
> A reader endpoint is not a load balancer. It is a DNS record that will resolve to an IP address of one of the replica nodes in a round robin fashion.

For read activity, applications can also connect to any node in the cluster. Unlike the primary endpoint, node endpoints resolve to specific endpoints. If you make a change in your cluster, such as adding or deleting a replica, you must update the node endpoints in your application.

### Valkey or Redis OSS (cluster mode enabled)

Valkey or Redis OSS (cluster mode enabled) clusters with replicas, because they have multiple shards (API/CLI: node groups), which mean they also have multiple primary nodes, have a different endpoint structure than Valkey or Redis OSS (cluster mode disabled) clusters. Valkey or Redis OSS (cluster mode enabled) has a *configuration endpoint* which "knows" all the primary and node endpoints in the cluster. Your application connects to the configuration endpoint. Whenever your

application writes to or reads from the cluster's configuration endpoint, Valkey and Redis OSS, behind the scenes, determine which shard the key belongs to and which endpoint in that shard to use. It is all quite transparent to your application.

You can find the endpoints for a cluster using the ElastiCache console, the AWS CLI, or the ElastiCache API.

**Finding Replication Group Endpoints**

To find the endpoints for your replication group, see one of the following topics:

- Finding a Valkey or Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding Endpoints for a Valkey or Redis OSS (Cluster Mode Enabled) Cluster (Console)
- Finding the Endpoints for Valkey or Redis OSS Replication Groups (AWS CLI)
- Finding Endpoints for Valkey or Redis OSS Replication Groups (ElastiCache API)

# Modifying a replication group

**⚠ Important Constraints**

- Currently, ElastiCache supports limited modifications of a Valkey or Redis OSS (cluster mode enabled) replication group, for example changing the engine version, using the API operation `ModifyReplicationGroup` (CLI: `modify-replication-group`). You can modify the number of shards (node groups) in a Valkey or Redis OSS (cluster mode enabled) cluster with the API operation `ModifyReplicationGroupShardConfiguration` (CLI: `modify-replication-group-shard-configuration`). For more information, see Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters.

  Other modifications to a Valkey or Redis OSS (cluster mode enabled) cluster require that you create a cluster with the new cluster incorporating the changes.
- You can upgrade Valkey or Redis OSS (cluster mode disabled) and Valkey or Redis OSS (cluster mode enabled) clusters and replication groups to newer engine versions. However, you can't downgrade to earlier engine versions except by deleting the existing cluster or replication group and creating it again. For more information, see Version Management for ElastiCache.
- You can upgrade an existing ElastiCache for Valkey or Redis OSS cluster that uses cluster mode disabled to use cluster mode enabled, using the console, ModifyReplicationGroup API or the modify-replication-group CLI command, as shown in the example below. Or you can follow the steps at Modifying cluster mode.

You can modify a Valkey or Redis OSS (cluster mode disabled) cluster's settings using the ElastiCache console, the AWS CLI, or the ElastiCache API. Currently, ElastiCache supports a limited number of modifications on a Valkey or Redis OSS (cluster mode enabled) replication group. Other modifications require you create a backup of the current replication group then using that backup to seed a new Valkey or Redis OSS (cluster mode enabled) replication group.

### Topics

- Using the AWS Management Console
- Using the AWS CLI
- Using the ElastiCache API

### Using the AWS Management Console

To modify a Valkey or Redis OSS (cluster mode disabled) cluster, see Modifying an ElastiCache cluster.

### Using the AWS CLI

The following are AWS CLI examples of the `modify-replication-group` command. You can use the same command to make other modifications to a replication group.

**Enable Multi-AZ on an existing Valkey or Redis OSS replication group:**

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
   --replication-group-id myReplGroup \
   --multi-az-enabled = true
```

For Windows:

```
aws elasticache modify-replication-group ^
   --replication-group-id myReplGroup ^
   --multi-az-enabled
```

**Modify cluster mode from disabled to enabled:**

To modify cluster mode from *disabled* to *enabled*, you must first set the cluster mode to *compatible*. Compatible mode allows your Valkey or Redis OSS clients to connect using both cluster mode enabled and cluster mode disabled. After you migrate all Valkey or Redis OSS clients to use cluster mode enabled, you can then complete cluster mode configuration and set the cluster mode to *enabled*.

For Linux, macOS, or Unix:

Set to cluster mode to *compatible*.

```
aws elasticache modify-replication-group \
   --replication-group-id myReplGroup \
   --cache-parameter-group-name myParameterGroupName \
   --cluster-mode compatible
```

Set to cluster mode to *enabled*.

```
aws elasticache modify-replication-group \
   --replication-group-id myReplGroup \
   --cluster-mode enabled
```

For Windows:

Set to cluster mode to *compatible*.

```
aws elasticache modify-replication-group ^
   --replication-group-id myReplGroup ^
   --cache-parameter-group-name myParameterGroupName ^
   --cluster-mode compatible
```

Set to cluster mode to *enabled*.

```
aws elasticache modify-replication-group ^
   --replication-group-id myReplGroup ^
   --cluster-mode enabled
```

For more information on the AWS CLI `modify-replication-group` command, see modify-replication-group or Modifying cluster mode in the *ElastiCache for Redis OSS User Guide*.

**Using the ElastiCache API**

The following ElastiCache API operation enables Multi-AZ on an existing Valkey or Redis OSS replication group. You can use the same operation to make other modifications to a replication group.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=ModifyReplicationGroup
   &AutomaticFailoverEnabled=true
   &Mutli-AZEnabled=true
   &ReplicationGroupId=myReplGroup
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20141201T220302Z
   &Version=2014-12-01
   &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
   &X-Amz-Date=20141201T220302Z
```

```
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information on the ElastiCache API `ModifyReplicationGroup` operation, see ModifyReplicationGroup.

## Deleting a replication group

If you no longer need one of your clusters with replicas (called *replication groups* in the API/CLI), you can delete it. When you delete a replication group, ElastiCache deletes all of the nodes in that group.

After you have begun this operation, it cannot be interrupted or canceled.

**Warning**

- When you delete an ElastiCache for Redis OSS cluster, your manual snapshots are retained. You will also have an option to create a final snapshot before the cluster is deleted. Automatic cache snapshots are not retained.
- `CreateSnapshot` permission is required to create a final snapshot. Without this permission, the API call will fail with an `Access Denied` exception.

### Deleting a Replication Group (Console)

To delete a cluster that has replicas, see Deleting a cluster in ElastiCache.

### Deleting a Replication Group (AWS CLI)

Use the command delete-replication-group to delete a replication group.

```
aws elasticache delete-replication-group --replication-group-id my-repgroup
```

A prompt asks you to confirm your decision. Enter *y* (yes) to start the operation immediately. After the process starts, it is irreversible.

```
  After you begin deleting this replication group, all of its nodes will be deleted as
well.
  Are you sure you want to delete this replication group? [Ny]y

REPLICATIONGROUP  my-repgroup  My replication group  deleting
```

### Deleting a replication group (ElastiCache API)

Call DeleteReplicationGroup with the `ReplicationGroup` parameter.

### Example

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DeleteReplicationGroup
   &ReplicationGroupId=my-repgroup
   &Version=2014-12-01
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20141201T220302Z
   &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
   &X-Amz-Date=20141201T220302Z
   &X-Amz-SignedHeaders=Host
   &X-Amz-Expires=20141201T220302Z
   &X-Amz-Credential=<credential>
   &X-Amz-Signature=<signature>
```

**Note**

If you set the `RetainPrimaryCluster` parameter to `true`, all of the read replicas will be deleted, but the primary cluster will be retained.

## Changing the number of replicas

You can dynamically increase or decrease the number of read replicas in your Valkey or Redis OSS replication group using the AWS Management Console, the AWS CLI, or the ElastiCache API. If your replication group is a Valkey or Redis OSS (cluster mode enabled) replication group, you can choose which shards (node groups) to increase or decrease the number of replicas.

To dynamically change the number of replicas in your replication group, choose the operation from the following table that fits your situation.

| To Do This | For Valkey or Redis OSS (cluster mode enabled) | For Valkey or Redis OSS (cluster mode disabled) |
|---|---|---|
| Add replicas | Increasing the number of replicas in a shard | Increasing the number of replicas in a shard<br><br>Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled) |
| Delete replicas | Decreasing the number of replicas in a shard | Decreasing the number of replicas in a shard<br><br>Deleting a read replica for Valkey or Redis OSS (Cluster Mode Disabled) |

### Increasing the number of replicas in a shard

You can increase the number of replicas in a Valkey or Redis OSS (cluster mode enabled) shard or Valkey or Redis OSS (cluster mode disabled) replication group up to a maximum of five. You can do so using the AWS Management Console, the AWS CLI, or the ElastiCache API.

**Topics**

- Using the AWS Management Console
- Using the AWS CLI
- Using the ElastiCache API

### Using the AWS Management Console

The following procedure uses the console to increase the number of replicas in a Valkey or Redis OSS (cluster mode enabled) replication group.

**To increase the number of replicas in shards**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**, and then choose the name of the replication group that you want to add replicas to.
3. Choose the box for each shard that you want to add replicas to.
4. Choose **Add replicas**.
5. Complete the **Add Replicas to Shards** page:

   - For **New number of replicas/shard**, enter the number of replicas that you want all of your selected shards to have. This value must be greater than or equal to **Current Number of Replicas per shard** and less than or equal to five. We recommend at least two replicas as a working minimum.
   - For **Availability Zones**, choose either **No preference** to have ElastiCache chose an Availability Zone for each new replica, or **Specify Availability Zones** to choose an Availability Zone for each new replica.

     If you choose **Specify Availability Zones**, for each new replica specify an Availability Zone using the list.
6. Choose **Add** to add the replicas or **Cancel** to cancel the operation.

### Using the AWS CLI

To increase the number of replicas in a Valkey or Redis OSS shard, use the `increase-replica-count` command with the following parameters:

- `--replication-group-id` – Required. Identifies which replication group you want to increase the number of replicas in.
- `--apply-immediately` or `--no-apply-immediately` – Required. Specifies whether to increase the replica count immediately (`--apply-immediately`) or at the next maintenance window (`--no-apply-immediately`). Currently, `--no-apply-immediately` is not supported.
- `--new-replica-count` – Optional. Specifies the number of replica nodes you want when finished, up to a maximum of five. Use this parameter for Valkey or Redis OSS (cluster mode disabled) replication groups where there is only one node group or Valkey or Redis OSS (cluster mode enabled) group, or where you want all node groups to have the same number of replicas. If this value is not larger than the current number of replicas in the node group, the call fails with an exception.
- `--replica-configuration` – Optional. Allows you to set the number of replicas and Availability Zones for each node group independently. Use this parameter for Valkey or Redis OSS (cluster mode enabled) groups where you want to configure each node group independently.

  `--replica-configuration` has three optional members:

  - `NodeGroupId` – The four-digit ID for the node group that you are configuring. For Valkey or Redis OSS (cluster mode disabled) replication groups, the shard ID is always `0001`. To find a Valkey or Redis OSS (cluster mode enabled) node group's (shard's) ID, see Finding a shard's ID.
  - `NewReplicaCount` – The number of replicas that you want in this node group at the end of this operation. The value must be more than the current number of replicas, up to a maximum of five. If this value is not larger than the current number of replicas in the node group, the call fails with an exception.
  - `PreferredAvailabilityZones` – A list of `PreferredAvailabilityZone` strings that specify which Availability Zones the replication group's nodes are to be in. The number of `PreferredAvailabilityZone` values must equal the value of `NewReplicaCount` plus 1 to account for the primary node. If this member of `--replica-configuration` is omitted, ElastiCache for Redis OSS chooses the Availability Zone for each of the new replicas.

> **Important**
>
> You must include either the `--new-replica-count` or `--replica-configuration` parameter, but not both, in your call.

### Example

The following example increases the number of replicas in the replication group `sample-repl-group` to three. When the example is finished, there are three replicas in each node group. This number applies whether this is a Valkey or Redis OSS (cluster mode disabled) group with a single node group or a Valkey or Redis OSS (cluster mode enabled) group with multiple node groups.

For Linux, macOS, or Unix:

```
aws elasticache increase-replica-count \
    --replication-group-id sample-repl-group \
    --new-replica-count 3 \
    --apply-immediately
```

For Windows:

```
aws elasticache increase-replica-count ^
    --replication-group-id sample-repl-group ^
    --new-replica-count 3 ^
    --apply-immediately
```

The following example increases the number of replicas in the replication group `sample-repl-group` to the value specified for the two specified node groups. Given that there are multiple node groups, this is a Valkey or Redis OSS (cluster mode enabled) replication group. When specifying the optional `PreferredAvailabilityZones`, the number of Availability Zones listed must equal the value of `NewReplicaCount` plus 1 more. This approach accounts for the primary node for the group identified by `NodeGroupId`.

For Linux, macOS, or Unix:

```
aws elasticache increase-replica-count \
    --replication-group-id sample-repl-group \
    --replica-configuration \
```

```
        NodeGroupId=0001,NewReplicaCount=2,PreferredAvailabilityZones=us-east-1a,us-
east-1c,us-east-1b \
        NodeGroupId=0003,NewReplicaCount=3,PreferredAvailabilityZones=us-east-1a,us-
east-1b,us-east-1c,us-east-1c \
    --apply-immediately
```

For Windows:

```
aws elasticache increase-replica-count ^
    --replication-group-id sample-repl-group ^
    --replica-configuration ^
        NodeGroupId=0001,NewReplicaCount=2,PreferredAvailabilityZones=us-east-1a,us-
east-1c,us-east-1b ^
        NodeGroupId=0003,NewReplicaCount=3,PreferredAvailabilityZones=us-east-1a,us-
east-1b,us-east-1c,us-east-1c \
    --apply-immediately
```

For more information about increasing the number of replicas using the CLI, see increase-replica-count in the *Amazon ElastiCache Command Line Reference.*

**Using the ElastiCache API**

To increase the number of replicas in a Valkey or Redis OSS shard, use the `IncreaseReplicaCount` action with the following parameters:

- `ReplicationGroupId` – Required. Identifies which replication group you want to increase the number of replicas in.
- `ApplyImmediately` – Required. Specifies whether to increase the replica count immediately (`ApplyImmediately=True`) or at the next maintenance window (`ApplyImmediately=False`). Currently, `ApplyImmediately=False` is not supported.
- `NewReplicaCount` – Optional. Specifies the number of replica nodes you want when finished, up to a maximum of five. Use this parameter for Valkey or Redis OSS (cluster mode disabled) replication groups where there is only one node group, or Valkey or Redis OSS (cluster mode enabled) groups where you want all node groups to have the same number of replicas. If this value is not larger than the current number of replicas in the node group, the call fails with an exception.
- `ReplicaConfiguration` – Optional. Allows you to set the number of replicas and Availability Zones for each node group independently. Use this parameter for Valkey or Redis OSS (cluster mode enabled) groups where you want to configure each node group independently.

`ReplicaConfiguraion` has three optional members:

- `NodeGroupId` – The four-digit ID for the node group you are configuring. For Valkey or Redis OSS (cluster mode disabled) replication groups, the node group (shard) ID is always `0001`. To find a Valkey or Redis OSS (cluster mode enabled) node group's (shard's) ID, see Finding a shard's ID.
- `NewReplicaCount` – The number of replicas that you want in this node group at the end of this operation. The value must be more than the current number of replicas and a maximum of five. If this value is not larger than the current number of replicas in the node group, the call fails with an exception.
- `PreferredAvailabilityZones` – A list of `PreferredAvailabilityZone` strings that specify which Availability Zones the replication group's nodes are to be in. The number of `PreferredAvailabilityZone` values must equal the value of `NewReplicaCount` plus 1 to account for the primary node. If this member of `ReplicaConfiguration` is omitted, ElastiCache for Redis OSS chooses the Availability Zone for each of the new replicas.

**Important**

You must include either the `NewReplicaCount` or `ReplicaConfiguration` parameter, but not both, in your call.

### Example

The following example increases the number of replicas in the replication group `sample-repl-group` to three. When the example is finished, there are three replicas in each node group. This number applies whether this is a Valkey or Redis OSS (cluster mode disabled) group with a single node group, or a Valkey or Redis OSS (cluster mode enabled) group with multiple node groups.

```
https://elasticache.us-west-2.amazonaws.com/
      ?Action=IncreaseReplicaCount
      &ApplyImmediately=True
      &NewReplicaCount=3
      &ReplicationGroupId=sample-repl-group
      &Version=2015-02-02
      &SignatureVersion=4
      &SignatureMethod=HmacSHA256
      &Timestamp=20150202T192317Z
```

```
      &X-Amz-Credential=<credential>
```

The following example increases the number of replicas in the replication group `sample-repl-group` to the value specified for the two specified node groups. Given that there are multiple node groups, this is a Valkey or Redis OSS (cluster mode enabled) replication group. When specifying the optional `PreferredAvailabilityZones`, the number of Availability Zones listed must equal the value of `NewReplicaCount` plus 1 more. This approach accounts for the primary node, for the group identified by `NodeGroupId`.

```
https://elasticache.us-west-2.amazonaws.com/
      ?Action=IncreaseReplicaCount
      &ApplyImmediately=True
      &ReplicaConfiguration.ConfigureShard.1.NodeGroupId=0001
      &ReplicaConfiguration.ConfigureShard.1.NewReplicaCount=2

 &ReplicaConfiguration.ConfigureShard.1.PreferredAvailabilityZones.PreferredAvailabilityZone.1=
east-1a

 &ReplicaConfiguration.ConfigureShard.1.PreferredAvailabilityZones.PreferredAvailabilityZone.2=
east-1c

 &ReplicaConfiguration.ConfigureShard.1.PreferredAvailabilityZones.PreferredAvailabilityZone.3=
east-1b
      &ReplicaConfiguration.ConfigureShard.2.NodeGroupId=0003
      &ReplicaConfiguration.ConfigureShard.2.NewReplicaCount=3

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.1=
east-1a

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.2=
east-1b

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.3=
east-1c

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.4=
east-1c
      &ReplicationGroupId=sample-repl-group
      &Version=2015-02-02
      &SignatureVersion=4
      &SignatureMethod=HmacSHA256
      &Timestamp=20150202T192317Z
```

```
        &X-Amz-Credential=<credential>
```

For more information about increasing the number of replicas using the API, see IncreaseReplicaCount in the *Amazon ElastiCache API Reference.*

### Decreasing the number of replicas in a shard

You can decrease the number of replicas in a shard for Valkey or Redis OSS (cluster mode enabled), or in a replication group for Valkey or Redis OSS (cluster mode disabled):

- For Valkey or Redis OSS (cluster mode disabled), you can decrease the number of replicas to one if Multi-AZ is enabled, and to zero if it isn't enabled.
- For Valkey or Redis OSS (cluster mode enabled), you can decrease the number of replicas to zero. However, you can't fail over to a replica if your primary node fails.

You can use the AWS Management Console, the AWS CLI or the ElastiCache API to decrease the number of replicas in a node group (shard) or replication group.

**Topics**

- Using the AWS Management Console
- Using the AWS CLI
- Using the ElastiCache API

#### Using the AWS Management Console

The following procedure uses the console to decrease the number of replicas in a Valkey or Redis OSS (cluster mode enabled) replication group.

**To decrease the number of replicas in a Valkey or Redis OSS shard**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Valkey** or **Redis OSS**, then choose the name of the replication group from which you want to delete replicas.
3. Choose the box for each shard you want to remove a replica node from.
4. Choose **Delete replicas**.
5. Complete the **Delete Replicas from Shards** page:

    a. For **New number of replicas/shard**, enter the number of replicas that you want the selected shards to have. This number must be greater than or equal to 1. We recommend at least two replicas per shard as a working minimum.

    b. Choose **Delete** to delete the replicas or **Cancel** to cancel the operation.

**Important**

- If you don't specify the replica nodes to be deleted, ElastiCache for Redis OSS automatically selects replica nodes for deletion. While doing so, ElastiCache for Redis OSS attempts to retain the Multi-AZ architecture for your replication group followed by retaining replicas with minimum replication lag with the primary.
- You can't delete the primary or primary nodes in a replication group. If you specify a primary node for deletion, the operation fails with an error event indicating that the primary node was selected for deletion.

### Using the AWS CLI

To decrease the number of replicas in a Valkey or Redis OSS shard, use the `decrease-replica-count` command with the following parameters:

- `--replication-group-id` – Required. Identifies which replication group you want to decrease the number of replicas in.
- `--apply-immediately` or `--no-apply-immediately` – Required. Specifies whether to decrease the replica count immediately (`--apply-immediately`) or at the next maintenance window (`--no-apply-immediately`). Currently, `--no-apply-immediately` is not supported.
- `--new-replica-count` – Optional. Specifies the number of replica nodes that you want. The value of `--new-replica-count` must be a valid value less than the current number of replicas in the node groups. For minimum permitted values, see Decreasing the number of replicas in a shard. If the value of `--new-replica-count` doesn't meet this requirement, the call fails.
- `--replicas-to-remove` – Optional. Contains a list of node IDs specifying the replica nodes to remove.
- `--replica-configuration` – Optional. Allows you to set the number of replicas and Availability Zones for each node group independently. Use this parameter for Valkey or Redis OSS (cluster mode enabled) groups where you want to configure each node group independently.

  `--replica-configuration` has three optional members:

  - `NodeGroupId` – The four-digit ID for the node group that you are configuring. For Valkey or Redis OSS (cluster mode disabled) replication groups, the shard ID is always `0001`. To find a Valkey or Redis OSS (cluster mode enabled) node group's (shard's) ID, see Finding a shard's ID.

- `NewReplicaCount` – An optional parameter that specifies the number of replica nodes you want. The value of `NewReplicaCount` must be a valid value less than the current number of replicas in the node groups. For minimum permitted values, see Decreasing the number of replicas in a shard. If the value of `NewReplicaCount` doesn't meet this requirement, the call fails.
- `PreferredAvailabilityZones` – A list of `PreferredAvailabilityZone` strings that specify which Availability Zones the replication group's nodes are in. The number of `PreferredAvailabilityZone` values must equal the value of `NewReplicaCount` plus 1 to account for the primary node. If this member of `--replica-configuration` is omitted, ElastiCache for Redis OSS chooses the Availability Zone for each of the new replicas.

> **Important**
>
> You must include one and only one of the `--new-replica-count`, `--replicas-to-remove`, or `--replica-configuration` parameters.

**Example**

The following example uses `--new-replica-count` to decrease the number of replicas in the replication group `sample-repl-group` to one. When the example is finished, there is one replica in each node group. This number applies whether this is a Valkey or Redis OSS (cluster mode disabled) group with a single node group or a Valkey or Redis OSS (cluster mode enabled) group with multiple node groups.

For Linux, macOS, or Unix:

```
aws elasticache decrease-replica-count
    --replication-group-id sample-repl-group \
    --new-replica-count 1 \
    --apply-immediately
```

For Windows:

```
aws elasticache decrease-replica-count ^
    --replication-group-id sample-repl-group ^
    --new-replica-count 1 ^
    --apply-immediately
```

The following example decreases the number of replicas in the replication group `sample-repl-group` by removing two specified replicas (`0001` and `0003`) from the node group.

For Linux, macOS, or Unix:

```
aws elasticache decrease-replica-count \
    --replication-group-id sample-repl-group \
    --replicas-to-remove 0001,0003 \
    --apply-immediately
```

For Windows:

```
aws elasticache decrease-replica-count ^
    --replication-group-id sample-repl-group ^
    --replicas-to-remove 0001,0003 \
    --apply-immediately
```

The following example uses `--replica-configuration` to decrease the number of replicas in the replication group `sample-repl-group` to the value specified for the two specified node groups. Given that there are multiple node groups, this is a Valkey or Redis OSS (cluster mode enabled) replication group. When specifying the optional `PreferredAvailabilityZones`, the number of Availability Zones listed must equal the value of `NewReplicaCount` plus 1 more. This approach accounts for the primary node for the group identified by `NodeGroupId`.

For Linux, macOS, or Unix:

```
aws elasticache decrease-replica-count \
    --replication-group-id sample-repl-group \
    --replica-configuration \
        NodeGroupId=0001,NewReplicaCount=1,PreferredAvailabilityZones=us-east-1a,us-east-1c \
        NodeGroupId=0003,NewReplicaCount=2,PreferredAvailabilityZones=us-east-1a,us-east-1b,us-east-1c \
    --apply-immediately
```

For Windows:

```
aws elasticache decrease-replica-count ^
    --replication-group-id sample-repl-group ^
    --replica-configuration ^
```

```
        NodeGroupId=0001,NewReplicaCount=2,PreferredAvailabilityZones=us-east-1a,us-
east-1c ^
        NodeGroupId=0003,NewReplicaCount=3,PreferredAvailabilityZones=us-east-1a,us-
east-1b,us-east-1c \
    --apply-immediately
```

For more information about decreasing the number of replicas using the CLI, see decrease-replica-count in the *Amazon ElastiCache Command Line Reference.*

**Using the ElastiCache API**

To decrease the number of replicas in a Valkey or Redis OSS shard, use the `DecreaseReplicaCount` action with the following parameters:

- `ReplicationGroupId` – Required. Identifies which replication group you want to decrease the number of replicas in.
- `ApplyImmediately` – Required. Specifies whether to decrease the replica count immediately (`ApplyImmediately=True`) or at the next maintenance window (`ApplyImmediately=False`). Currently, `ApplyImmediately=False` is not supported.
- `NewReplicaCount` – Optional. Specifies the number of replica nodes you want. The value of `NewReplicaCount` must be a valid value less than the current number of replicas in the node groups. For minimum permitted values, see Decreasing the number of replicas in a shard. If the value of `--new-replica-count` doesn't meet this requirement, the call fails.
- `ReplicasToRemove` – Optional. Contains a list of node IDs specifying the replica nodes to remove.
- `ReplicaConfiguration` – Optional. Contains a list of node groups that allows you to set the number of replicas and Availability Zones for each node group independently. Use this parameter for Valkey or Redis OSS (cluster mode enabled) groups where you want to configure each node group independently.

  `ReplicaConfiguraion` has three optional members:

  - `NodeGroupId` – The four-digit ID for the node group you are configuring. For Valkey or Redis OSS (cluster mode disabled) replication groups, the node group ID is always `0001`. To find a Valkey or Redis OSS (cluster mode enabled) node group's (shard's) ID, see Finding a shard's ID.
  - `NewReplicaCount` – The number of replicas that you want in this node group at the end of this operation. The value must be less than the current number of replicas down to a minimum of 1 if Multi-AZ is enabled or 0 if Multi-AZ with Automatic Failover isn't enabled. If this value is not less than the current number of replicas in the node group, the call fails with an exception.

- `PreferredAvailabilityZones` – A list of `PreferredAvailabilityZone` strings that specify which Availability Zones the replication group's nodes are in. The number of `PreferredAvailabilityZone` values must equal the value of `NewReplicaCount` plus 1 to account for the primary node. If this member of `ReplicaConfiguration` is omitted, ElastiCache for Redis OSS chooses the Availability Zone for each of the new replicas.

> **Important**
>
> You must include one and only one of the `NewReplicaCount`, `ReplicasToRemove`, or `ReplicaConfiguration` parameters.

### Example

The following example uses `NewReplicaCount` to decrease the number of replicas in the replication group `sample-repl-group` to one. When the example is finished, there is one replica in each node group. This number applies whether this is a Valkey or Redis OSS (cluster mode disabled) group with a single node group or a Valkey or Redis OSS (cluster mode enabled) group with multiple node groups.

```
https://elasticache.us-west-2.amazonaws.com/
      ?Action=DecreaseReplicaCount
      &ApplyImmediately=True
      &NewReplicaCount=1
      &ReplicationGroupId=sample-repl-group
      &Version=2015-02-02
      &SignatureVersion=4
      &SignatureMethod=HmacSHA256
      &Timestamp=20150202T192317Z
      &X-Amz-Credential=<credential>
```

The following example decreases the number of replicas in the replication group `sample-repl-group` by removing two specified replicas (`0001` and `0003`) from the node group.

```
https://elasticache.us-west-2.amazonaws.com/
      ?Action=DecreaseReplicaCount
      &ApplyImmediately=True
      &ReplicasToRemove.ReplicaToRemove.1=0001
      &ReplicasToRemove.ReplicaToRemove.2=0003
```

```
      &ReplicationGroupId=sample-repl-group
      &Version=2015-02-02
      &SignatureVersion=4
      &SignatureMethod=HmacSHA256
      &Timestamp=20150202T192317Z
      &X-Amz-Credential=<credential>
```

The following example uses `ReplicaConfiguration` to decrease the number of replicas in the replication group `sample-repl-group` to the value specified for the two specified node groups. Given that there are multiple node groups, this is a Valkey or Redis OSS (cluster mode enabled) replication group. When specifying the optional `PreferredAvailabilityZones`, the number of Availability Zones listed must equal the value of `NewReplicaCount` plus 1 more. This approach accounts for the primary node for the group identified by `NodeGroupId`.

```
https://elasticache.us-west-2.amazonaws.com/
      ?Action=DecreaseReplicaCount
      &ApplyImmediately=True
      &ReplicaConfiguration.ConfigureShard.1.NodeGroupId=0001
      &ReplicaConfiguration.ConfigureShard.1.NewReplicaCount=1

 &ReplicaConfiguration.ConfigureShard.1.PreferredAvailabilityZones.PreferredAvailabilityZone.1=
east-1a

 &ReplicaConfiguration.ConfigureShard.1.PreferredAvailabilityZones.PreferredAvailabilityZone.2=
east-1c
      &ReplicaConfiguration.ConfigureShard.2.NodeGroupId=0003
      &ReplicaConfiguration.ConfigureShard.2.NewReplicaCount=2

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.1=
east-1a

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.2=
east-1b

 &ReplicaConfiguration.ConfigureShard.2.PreferredAvailabilityZones.PreferredAvailabilityZone.4=
east-1c
      &ReplicationGroupId=sample-repl-group
      &Version=2015-02-02
      &SignatureVersion=4
      &SignatureMethod=HmacSHA256
      &Timestamp=20150202T192317Z
      &X-Amz-Credential=<credential>
```

For more information about decreasing the number of replicas using the API, see DecreaseReplicaCount in the *Amazon ElastiCache API Reference.*

**Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled)**

Information in the following topic applies to Valkey or Redis OSS (cluster mode disabled) replication groups only.

As your read traffic increases, you might want to spread those reads across more nodes and reduce the read pressure on any one node. In this topic, you can find how to add a read replica to a Valkey or Redis OSS (cluster mode disabled) cluster.

A Valkey or Redis OSS (cluster mode disabled) replication group can have a maximum of five read replicas. If you attempt to add a read replica to a replication group that already has five read replicas, the operation fails.

For information about adding replicas to a Valkey or Redis OSS (cluster mode enabled) replication group, see the following:

- Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters
- Increasing the number of replicas in a shard

You can add a read replica to a Valkey or Redis OSS (cluster mode disabled) cluster using the ElastiCache Console, the AWS CLI, or the ElastiCache API.

**Related topics**

- Adding nodes to an ElastiCache cluster
- Adding a read replica to a replication group (AWS CLI)
- Adding a read replica to a replication group using the API

**Adding a read replica to a replication group (AWS CLI)**

To add a read replica to a Valkey or Redis OSS (cluster mode disabled) replication group, use the AWS CLI `create-cache-cluster` command, with the parameter `--replication-group-id` to specify which replication group to add the cluster (node) to.

The following example creates the cluster `my-read replica` and adds it to the replication group `my-replication-group`. The node types, parameter groups, security groups, maintenance

window, and other settings for the read replica are the same as for the other nodes in `my-replication-group`.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-cluster \
      --cache-cluster-id my-read-replica \
      --replication-group-id my-replication-group
```

For Windows:

```
aws elasticache create-cache-cluster ^
      --cache-cluster-id my-read-replica ^
      --replication-group-id my-replication-group
```

For more information on adding a read replica using the CLI, see create-cache-cluster in the *Amazon ElastiCache Command Line Reference.*

**Adding a read replica to a replication group using the API**

To add a read replica to a Valkey or Redis OSS (cluster mode disabled) replication group, use the ElastiCache `CreateCacheCluster` operation, with the parameter `ReplicationGroupId` to specify which replication group to add the cluster (node) to.

The following example creates the cluster `myReadReplica` and adds it to the replication group `myReplicationGroup`. The node types, parameter groups, security groups, maintenance window, and other settings for the read replica are the same as for the other nodes `myReplicationGroup`.

```
https://elasticache.us-west-2.amazonaws.com/
      ?Action=CreateCacheCluster
      &CacheClusterId=myReadReplica
      &ReplicationGroupId=myReplicationGroup
      &Version=2015-02-02
      &SignatureVersion=4
      &SignatureMethod=HmacSHA256
      &Timestamp=20150202T192317Z
      &X-Amz-Credential=<credential>
```

For more information on adding a read replica using the API, see CreateCacheCluster in the *Amazon ElastiCache API Reference.*

### Deleting a read replica for Valkey or Redis OSS (Cluster Mode Disabled)

Information in the following topic applies to Valkey or Redis OSS (cluster mode disabled) replication groups only.

As read traffic on your Valkey or Redis OSS replication group changes, you might want to add or remove read replicas. Removing a node from a replication group is the same as just deleting a cluster, though there are restrictions:

- You cannot remove the primary from a replication group. If you want to delete the primary, do the following:

  1. Promote a read replica to primary. For more information on promoting a read replica to primary, see Promoting a read replica to primary, for Valkey or Redis OSS (cluster mode disabled) replication groups.
  2. Delete the old primary. For a restriction on this method, see the next point.
- If Multi-AZ is enabled on a replication group, you can't remove the last read replica from the replication group. In this case, do the following:

  1. Modify the replication group by disabling Multi-AZ. For more information, see Modifying a replication group.
  2. Delete the read replica.

You can remove a read replica from a Valkey or Redis OSS (cluster mode disabled) replication group using the ElastiCache console, the AWS CLI for ElastiCache, or the ElastiCache API.

For directions on deleting a cluster from a Valkey or Redis OSS replication group, see the following:

- Using the AWS Management Console
- Using the AWS CLI to delete an ElastiCache cluster
- Using the ElastiCache API
- Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters
- Decreasing the number of replicas in a shard

## Promoting a read replica to primary, for Valkey or Redis OSS (cluster mode disabled) replication groups

Information in the following topic applies to only Valkey or Redis OSS (cluster mode disabled) replication groups.

You can promote a Valkey or Redis OSS (cluster mode disabled) read replica to primary using the AWS Management Console, the AWS CLI, or the ElastiCache API. You can't promote a read replica to primary while Multi-AZ with Automatic Failover is enabled on the replication group. To promote a Valkey or Redis OSS (cluster mode disabled) replica to primary on a Multi-AZ enabled replication group, do the following:

1. Modify the replication group to disable Multi-AZ (doing this doesn't require that all your clusters be in the same Availability Zone). For more information, see Modifying a replication group.
2. Promote the read replica to primary.
3. Modify the replication group to re-enable Multi-AZ.

### Using the AWS Management Console

The following procedure uses the console to promote a replica node to primary.

**To promote a read replica to primary (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. If the replica you want to promote is a member of a Valkey or Redis OSS (cluster mode disabled) replication group where Multi-AZ is enabled, modify the replication group to disable Multi-AZ before you proceed. For more information, see Modifying a replication group.
3. Choose **Valkey** or **Redis OSS**, then from the list of clusters, choose the replication group that you want to modify. This replication group must be running the "Redis" engine, not the "Clustered Redis" engine, and must have two or more nodes.
4. From the list of nodes, choose the replica node you want to promote to primary, then for **Actions**, choose **Promote**.
5. In the **Promote Read Replica** dialog box, do the following:

a. For **Apply Immediately**, choose **Yes** to promote the read replica immediately, or **No** to promote it at the cluster's next maintenance window.

b. Choose **Promote** to promote the read replica or **Cancel** to cancel the operation.

6. If the cluster was Multi-AZ enabled before you began the promotion process, wait until the replication group's status is **available**, then modify the cluster to re-enable Multi-AZ. For more information, see Modifying a replication group.

### Using the AWS CLI

You can't promote a read replica to primary if the replication group is Multi-AZ enabled. In some cases, the replica that you want to promote might be a member of a replication group where Multi-AZ is enabled. In these cases, you must modify the replication group to disable Multi-AZ before you proceed. Doing this doesn't require that all your clusters be in the same Availability Zone. For more information on modifying a replication group, see Modifying a replication group.

The following AWS CLI command modifies the replication group `sample-repl-group`, making the read replica `my-replica-1` the primary in the replication group.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
   --replication-group-id sample-repl-group \
   --primary-cluster-id my-replica-1
```

For Windows:

```
aws elasticache modify-replication-group ^
   --replication-group-id sample-repl-group ^
   --primary-cluster-id my-replica-1
```

For more information on modifying a replication group, see modify-replication-group in the *Amazon ElastiCache Command Line Reference.*

### Using the ElastiCache API

You can't promote a read replica to primary if the replication group is Multi-AZ enabled. In some cases, the replica that you want to promote might be a member of a replication group where Multi-AZ is enabled. In these cases, you must modify the replication group to disable Multi-AZ before you

proceed. Doing this doesn't require that all your clusters be in the same Availability Zone. For more information on modifying a replication group, see Modifying a replication group.

The following ElastiCache API action modifies the replication group `myReplGroup`, making the read replica `myReplica-1` the primary in the replication group.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=ModifyReplicationGroup
   &ReplicationGroupId=myReplGroup
   &PrimaryClusterId=myReplica-1
   &Version=2014-12-01
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20141201T220302Z
   &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
   &X-Amz-Date=20141201T220302Z
   &X-Amz-SignedHeaders=Host
   &X-Amz-Expires=20141201T220302Z
   &X-Amz-Credential=<credential>
   &X-Amz-Signature=<signature>
```

For more information on modifying a replication group, see ModifyReplicationGroup in the *Amazon ElastiCache API Reference.*

## Managing ElastiCache cluster maintenance

Every cluster has a weekly maintenance window during which any system changes are applied. With Valkey and Redis OSS, replication groups have this same weekly maintenance window. If you don't specify a preferred maintenance window when you create or modify a cluster or replication group, ElastiCache assigns a 60-minute maintenance window within your region's maintenance window on a randomly chosen day of the week.

The 60-minute maintenance window is chosen at random from an 8-hour block of time per region. The following table lists the time blocks for each region from which the default maintenance windows are assigned. You may choose a preferred maintenance window outside the region's maintenance window block.

| Region Code | Region Name | Region Maintenance Window |
|---|---|---|
| `af-south-1` | Africa (Cape Town) Region | 13:00–21:00 UTC |

| Region Code | Region Name | Region Maintenance Window |
|---|---|---|
| ap-east-1 | Asia Pacific (Hong Kong) Region | 13:00–21:00 UTC |
| ap-east-2 | Asia Pacific (Taipei) Region | 09:00–17:00 UTC |
| ap-northeast-1 | Asia Pacific (Tokyo) Region | 13:00–21:00 UTC |
| ap-northeast-2 | Asia Pacific (Seoul) Region | 12:00–20:00 UTC |
| ap-northeast-3 | Asia Pacific (Osaka) Region | 12:00–20:00 UTC |
| ap-south-1 | Asia Pacific (Mumbai) Region | 17:30–1:30 UTC |
| ap-south-2 | Asia Pacific (Hyderabad) Region | 06:30–14:30 UTC |
| ap-southeast-1 | Asia Pacific (Singapore) Region | 14:00–22:00 UTC |
| ap-southeast-2 | Asia Pacific (Sydney) Region | 12:00–20:00 UTC |
| ap-southeast-3 | Asia Pacific (Jakarta) Region | 14:00–22:00 UTC |
| ap-southeast-4 | Asia Pacific (Melbourne) Region | 11:00–19:00 UTC |
| ap-southeast-5 | Asia Pacific (Malaysia) Region | 09:00–17:00 UTC |
| ap-southeast-7 | Asia Pacific (Thailand) Region | 08:00–16:00 UTC |
| ca-central-1 | Canada (Central) Region | 03:00–11:00 UTC |
| ca-west-1 | Canada West (Calgary) Region | 18:00–02:00 UTC |
| cn-north-1 | China (Beijing) Region | 14:00–22:00 UTC |
| cn-northwest-1 | China (Ningxia) Region | 14:00–22:00 UTC |
| eu-central-1 | Europe (Frankfurt) Region | 23:00–07:00 UTC |
| eu-central-2 | Europe (Zurich) Region | 02:00–10:00 UTC |
| eu-north-1 | Europe (Stockholm) Region | 23:00–07:00 UTC |

| Region Code | Region Name | Region Maintenance Window |
|---|---|---|
| eu-south-1 | Europe (Milan) Region | 21:00–05:00 UTC |
| eu-south-2 | Europe (Spain) Region | 02:00–10:00 UTC |
| eu-west-1 | Europe (Ireland) Region | 22:00–06:00 UTC |
| eu-west-2 | Europe (London) Region | 23:00–07:00 UTC |
| eu-west-3 | EU (Paris) Region | 23:59–07:29 UTC |
| il-central-1 | Israel (Tel Aviv) Region | 03:00–11:00 UTC |
| me-central-1 | Middle East (UAE) Region | 13:00–21:00 UTC |
| me-south-1 | Middle East (Bahrain) Region | 13:00–21:00 UTC |
| mx-central-1 | Mexico (Central) Region | 19:00–03:00 UTC |
| sa-east-1 | South America (São Paulo) Region | 01:00–09:00 UTC |
| us-east-1 | US East (N. Virginia) Region | 03:00–11:00 UTC |
| us-east-2 | US East (Ohio) Region | 04:00–12:00 UTC |
| us-gov-east-1 | AWS GovCloud (US-East) | 17:00–01:00 UTC |
| us-gov-west-1 | AWS GovCloud (US) region | 06:00–14:00 UTC |
| us-west-1 | US West (N. California) Region | 06:00–14:00 UTC |
| us-west-2 | US West (Oregon) Region | 06:00–14:00 UTC |

### Changing your cluster's or replication group's maintenance window

The maintenance window should fall at the time of lowest usage and thus might need modification from time to time. You can modify your cluster or replication group to specify a time range of up to 24 hours in duration during which any maintenance activities you have requested should occur. Any deferred or pending cluster modifications you requested occur during this time.

> **Note**
>
> If you want to apply node type modifications and/or engine upgrades immediately using the AWS Management Console select the **Apply now** box. Otherwise these modifications will be applied during your next scheduled maintenance window. To the use the API, see modify-replication-group or modify-cache-cluster.

**More information**

For information on your maintenance window and node replacement, see the following:

- ElastiCache Maintenance—FAQ on maintenance and node replacement
- Replacing nodes (Memcached)—Managing node replacement for Memcached
- Modifying an ElastiCache cluster—Changing a cluster's maintenance window
- Replacing nodes (Valkey and Redis OSS)—Managing node replacement
- Modifying a replication group—Changing a replication group's maintenance window

## Configuring engine parameters using ElastiCache parameter groups

Amazon ElastiCache uses parameters to control the runtime properties of your nodes and clusters. Generally, newer engine versions include additional parameters to support the newer functionality. For tables of Memcached parameters, see Memcached specific parameters. For tables of Valkey and Redis OSS parameters, see Valkey and Redis OSS parameters.

As you would expect, some parameter values, such as `maxmemory`, are determined by the engine and node type. For a table of these Memcached parameter values by node type, see Memcached node-type specific parameters. For a table of these Valkey and Redis OSS parameter values by node type, see Redis OSS node-type specific parameters.

> **Note**
>
> For a list of Memcached-specific parameters, see Memcached Specific Parameters.

**Topics**

- Parameter management in ElastiCache

- Cache parameter group tiers in ElastiCache
- Creating an ElastiCache parameter group
- Listing ElastiCache parameter groups by name
- Listing an ElastiCache parameter group's values
- Modifying an ElastiCache parameter group
- Deleting an ElastiCache parameter group
- Engine specific parameters

## Parameter management in ElastiCache

ElastiCache parameters are grouped together into named parameter groups for easier parameter management. A parameter group represents a combination of specific values for the parameters that are passed to the engine software during startup. These values determine how the engine processes on each node behave at runtime. The parameter values on a specific parameter group apply to all nodes that are associated with the group, regardless of which cluster they belong to.

To fine-tune your cluster's performance, you can modify some parameter values or change the cluster's parameter group.

- You cannot modify or delete the default parameter groups. If you need custom parameter values, you must create a custom parameter group.
- For Memcached, the parameter group family and the cluster you're assigning it to must be compatible. For example, if your cluster is running Memcached version 1.4.8, you can only use parameter groups, default or custom, from the Memcached 1.4 family.

  For Redis OSS, the parameter group family and the cluster you're assigning it to must be compatible. For example, if your cluster is running Redis OSS version 7.1, you can only use parameter groups, default or custom, from the Redis OSS 7 family.
- If you change a cluster's parameter group, the values for any conditionally modifiable parameter must be the same in both the current and new parameter groups.
- For Memcached, when you change a cluster's parameters the change is applied to the cluster immediately. This is true whether you change the cluster's parameter group itself or a parameter value within the cluster's parameter group. To determine when a particular parameter change is applied, see the **Changes Take Effect** column in the tables for Memcached specific parameters. For information on rebooting a cluster's nodes, see Rebooting clusters.
- For Redis OSS, when you change a cluster's parameters, the change is applied to the cluster either immediately or, with the exceptions noted following, after the cluster nodes are rebooted. This is true whether you change the cluster's parameter group itself or a parameter value within the cluster's parameter group. To determine when a particular parameter change is applied, see the **Changes Take Effect** column in the tables for Valkey and Redis OSS parameters.

  For more information on rebooting Valkey or Redis OSS nodes, see Rebooting nodes.

**Valkey or Redis OSS (Cluster Mode Enabled) parameter changes**

If you make changes to the following parameters on a Valkey or Redis OSS (cluster mode enabled) cluster, follow the ensuing steps.

- activerehashing
- databases

1. Create a manual backup of your cluster. See Taking manual backups.
2. Delete the cluster. See Deleting clusters.
3. store the cluster using the altered parameter group and backup to seed the new cluster. See Restoring from a backup into a new cache.

Changes to other parameters do not require this.

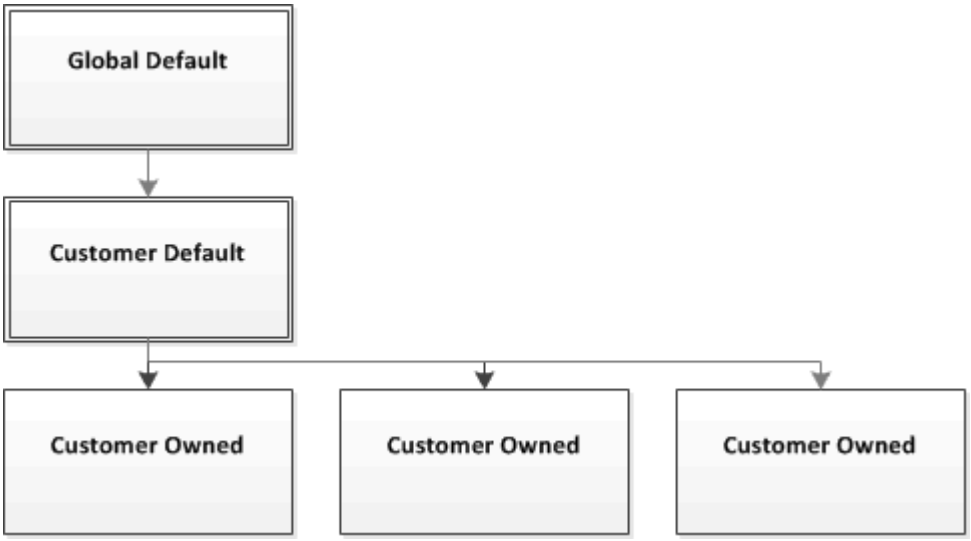
- You can associate parameter groups with Valkey and Redis OSS global datastores. *Global datastores* are a collection of one or more clusters that span AWS Regions. In this case, the parameter group is shared by all clusters that make up the global datastore. Any modifications to the parameter group of the primary cluster are replicated to all remaining clusters in the global datastore. For more information, see Replication across AWS Regions using global datastores.

  You can check if a parameter group is part of a global datastore by looking in these locations:

  - On the ElastiCache console on the **Parameter Groups** page, the yes/no **Global** attribute
  - The yes/no `IsGlobal` property of the CacheParameterGroup API operation

## Cache parameter group tiers in ElastiCache

Amazon ElastiCache has three tiers of cache parameter groups as shown following.




*Amazon ElastiCache parameter group tiers*

### Global Default

The top-level root parameter group for all Amazon ElastiCache customers in the region.

The global default cache parameter group:

- Is reserved for ElastiCache and not available to the customer.

### Customer Default

A copy of the Global Default cache parameter group which is created for the customer's use.

The Customer Default cache parameter group:

- Is created and owned by ElastiCache.
- Is available to the customer for use as a cache parameter group for any clusters running an engine version supported by this cache parameter group.
- Cannot be edited by the customer.

### Customer Owned

A copy of the Customer Default cache parameter group. A Customer Owned cache parameter group is created whenever the customer creates a cache parameter group.

The Customer Owned cache parameter group:

- Is created and owned by the customer.
- Can be assigned to any of the customer's compatible clusters.
- Can be modified by the customer to create a custom cache parameter group.

  Not all parameter values can be modified. For more information on Memcached values, see Memcached specific parameters. For more information on Valkey and Redis OSS values, see Valkey and Redis OSS parameters.

## Creating an ElastiCache parameter group

You need to create a new parameter group if there is one or more parameter values that you want changed from the default values. You can create a parameter group using the ElastiCache console, the AWS CLI, or the ElastiCache API.

### Creating an ElastiCache parameter group (Console)

The following procedure shows how to create a parameter group using the ElastiCache console.

**To create a parameter group using the ElastiCache console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of all available parameter groups, in the left hand navigation pane choose **Parameter Groups**.
3. To create a parameter group, choose **Create Parameter Group**.

   The **Create Parameter Group** screen appears.
4. From the **Family** list, choose the parameter group family that will be the template for your parameter group.

   The parameter group family, such as *memcached1.6* or *redis7* defines the actual parameters in your parameter group and their initial values. The parameter group family must coincide with the cluster's engine and version.
5. In the **Name** box, type in a unique name for this parameter group.

When creating a cluster or modifying a cluster's parameter group, you will choose the parameter group by its name. Therefore, we recommend that the name be informative and somehow identify the parameter group's family.

Parameter group naming constraints are as follows:

- Must begin with an ASCII letter.
- Can only contain ASCII letters, digits, and hyphens.
- Must be 1–255 characters long.
- Can't contain two consecutive hyphens.
- Can't end with a hyphen.

6. In the **Description** box, type in a description for the parameter group.
7. To create the parameter group, choose **Create**.

   To terminate the process without creating the parameter group, choose **Cancel**.
8. When the parameter group is created, it will have the family's default values. To change the default values you must modify the parameter group. For more information, see Modifying an ElastiCache parameter group.

### Creating an ElastiCache parameter group (AWS CLI)

To create a parameter group using the AWS CLI, use the command `create-cache-parameter-group` with these parameters.

- `--cache-parameter-group-name` — The name of the parameter group.

  Parameter group naming constraints are as follows:
  - Must begin with an ASCII letter.
  - Can only contain ASCII letters, digits, and hyphens.
  - Must be 1–255 characters long.
  - Can't contain two consecutive hyphens.
  - Can't end with a hyphen.
- `--cache-parameter-group-family` — The engine and version family for the parameter group.
- `--description` — A user supplied description for the parameter group.

**Example**

The following example creates a parameter group named *myMem14* using the memcached1.4 family as the template.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-parameter-group \
    --cache-parameter-group-name myMem14  \
    --cache-parameter-group-family memcached1.4 \
    --description "My first parameter group"
```

For Windows:

```
aws elasticache create-cache-parameter-group ^
    --cache-parameter-group-name myMem14  ^
    --cache-parameter-group-family memcached1.4 ^
    --description "My first parameter group"
```

The output from this command should look something like this.

```
{
    "CacheParameterGroup": {
        "CacheParameterGroupName": "myMem14",
        "CacheParameterGroupFamily": "memcached1.4",
        "Description": "My first  parameter group"
    }
}
```

**Example**

The following example creates a parameter group named *myRed7* using the redis7 family as the template.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-parameter-group \
    --cache-parameter-group-name myRed7  \
    --cache-parameter-group-family redis7 \
    --description "My first parameter group"
```

For Windows:

```
aws elasticache create-cache-parameter-group ^
    --cache-parameter-group-name myRed7  ^
    --cache-parameter-group-family redis7 ^
    --description "My first parameter group"
```

The output from this command should look something like this.

```
{
    "CacheParameterGroup": {
        "CacheParameterGroupName": "myRed7",
        "CacheParameterGroupFamily": "redis7",
        "Description": "My first parameter group"
    }
}
```

When the parameter group is created, it will have the family's default values. To change the default values you must modify the parameter group. For more information, see Modifying an ElastiCache parameter group.

For more information, see `create-cache-parameter-group`.

**Creating an ElastiCache parameter group (ElastiCache API)**

To create a parameter group using the ElastiCache API, use the `CreateCacheParameterGroup` action with these parameters.

- `ParameterGroupName` — The name of the parameter group.

  Parameter group naming constraints are as follows:

  - Must begin with an ASCII letter.
  - Can only contain ASCII letters, digits, and hyphens.
  - Must be 1–255 characters long.
  - Can't contain two consecutive hyphens.
  - Can't end with a hyphen.
- `CacheParameterGroupFamily` — The engine and version family for the parameter group. For example, `memcached1.4`.
- `CacheParameterGroupFamily` — The engine and version family for the parameter group. For example, `redis7`.

- `Description` — A user supplied description for the parameter group.

### Example

The following example creates a parameter group named *myMem14* using the memcached1.4 family as the template.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=CreateCacheParameterGroup
   &CacheParameterGroupFamily=memcached1.4
   &CacheParameterGroupName=myMem14
   &Description=My%20first%20parameter%20group
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action should look something like this.

```
<CreateCacheParameterGroupResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <CreateCacheParameterGroupResult>
    <CacheParameterGroup>
      <CacheParameterGroupName>myMem14</CacheParameterGroupName>
      <CacheParameterGroupFamily>memcached1.4</CacheParameterGroupFamily>
      <Description>My first  parameter group</Description>
    </CacheParameterGroup>
  </CreateCacheParameterGroupResult>
  <ResponseMetadata>
    <RequestId>d8465952-af48-11e0-8d36-859edca6f4b8</RequestId>
  </ResponseMetadata>
</CreateCacheParameterGroupResponse>
```

### Example

The following example creates a parameter group named *myRed7* using the redis7 family as the template.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=CreateCacheParameterGroup
```

```
   &CacheParameterGroupFamily=redis7
   &CacheParameterGroupName=myRed7
   &Description=My%20first%20parameter%20group
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action should look something like this.

```
<CreateCacheParameterGroupResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <CreateCacheParameterGroupResult>
    <CacheParameterGroup>
      <CacheParameterGroupName>myRed7</CacheParameterGroupName>
      <CacheParameterGroupFamily>redis7</CacheParameterGroupFamily>
      <Description>My first parameter group</Description>
    </CacheParameterGroup>
  </CreateCacheParameterGroupResult>
  <ResponseMetadata>
    <RequestId>d8465952-af48-11e0-8d36-859edca6f4b8</RequestId>
  </ResponseMetadata>
</CreateCacheParameterGroupResponse>
```

When the parameter group is created, it will have the family's default values. To change the default values you must modify the parameter group. For more information, see Modifying an ElastiCache parameter group.

For more information, see CreateCacheParameterGroup.

## Listing ElastiCache parameter groups by name

You can list the parameter groups using the ElastiCache console, the AWS CLI, or the ElastiCache API.

### Listing parameter groups by name (Console)

The following procedure shows how to view a list of the parameter groups using the ElastiCache console.

### To list parameter groups using the ElastiCache console

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of all available parameter groups, in the left hand navigation pane choose **Parameter Groups**.

### Listing ElastiCache parameter groups by name (AWS CLI)

To generate a list of parameter groups using the AWS CLI, use the command `describe-cache-parameter-groups`. If you provide a parameter group's name, only that parameter group will be listed. If you do not provide a parameter group's name, up to `--max-records` parameter groups will be listed. In either case, the parameter group's name, family, and description are listed.

### Example

The following sample code lists the parameter group *myMem14*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameter-groups \
    --cache-parameter-group-name myMem14
```

For Windows:

```
aws elasticache describe-cache-parameter-groups ^
    --cache-parameter-group-name myMem14
```

The output of this command will look something like this, listing the name, family, and description for the parameter group.

```
{
    "CacheParameterGroups": [
     {
         "CacheParameterGroupName": "myMem14",
         "CacheParameterGroupFamily": "memcached1.4",
         "Description": "My first parameter group"
     }
    ]
}
```

**Example**

The following sample code lists the parameter group *myRed7*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameter-groups \
    --cache-parameter-group-name myRed7
```

For Windows:

```
aws elasticache describe-cache-parameter-groups ^
    --cache-parameter-group-name myRed7
```

The output of this command will look something like this, listing the name, family, and description for the parameter group.

```
{
    "CacheParameterGroups": [
     {
         "CacheParameterGroupName": "myRed7",
         "CacheParameterGroupFamily": "redis7",
         "Description": "My first parameter group"
     }
    ]
}
```

**Example**

The following sample code lists the parameter group *myRed56* for parameter groups running on Redis OSS engine version 5.0.6 onwards. If the parameter group is part of a Replication across AWS Regions using global datastores, the `IsGlobal` property value returned in the output will be `Yes`.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameter-groups \
    --cache-parameter-group-name myRed56
```

For Windows:

```
aws elasticache describe-cache-parameter-groups ^
    --cache-parameter-group-name myRed56
```

The output of this command will look something like this, listing the name, family, isGlobal and description for the parameter group.

```
{
    "CacheParameterGroups": [
     {
         "CacheParameterGroupName": "myRed56",
         "CacheParameterGroupFamily": "redis5.0",
         "Description": "My first parameter group",
         "IsGlobal": "yes"
     }
    ]
}
```

**Example**

The following sample code lists up to 10 parameter groups.

```
aws elasticache describe-cache-parameter-groups --max-records 10
```

The JSON output of this command will look something like this, listing the name, family, description and, in the case of redis5.6 whether the parameter group is part of a global datastore (isGlobal), for each parameter group.

```
{
    "CacheParameterGroups": [
        {
            "CacheParameterGroupName": "custom-redis7",
            "CacheParameterGroupFamily": "redis7",
            "Description": "custom parameter group with reserved-memory > 0"
        },
        {
            "CacheParameterGroupName": "default.memcached1.6",
            "CacheParameterGroupFamily": "memcached1.6",
            "Description": "Default parameter group for memcached1.6"
        },
        {
            "CacheParameterGroupName": "default.redis6.x",
            "CacheParameterGroupFamily": "redis6.x",
            "Description": "Default parameter group for redis6.x"
        },
        {
            "CacheParameterGroupName": "default.redis7",
            "CacheParameterGroupFamily": "redis7",
            "Description": "Default parameter group for redis7"
        },
        {
            "CacheParameterGroupName": "default.redis7.cluster.on",
            "CacheParameterGroupFamily": "redis7",
            "Description": "Customized default parameter group for redis7 with cluster
 mode on",
            "isGlobal": "yes"
        }
    ]
}
```

For more information, see [describe-cache-parameter-groups](describe-cache-parameter-groups).

**Listing ElastiCache parameter groups by name (ElastiCache API)**

To generate a list of parameter groups using the ElastiCache API, use the `DescribeCacheParameterGroups` action. If you provide a parameter group's name, only that parameter group will be listed. If you do not provide a parameter group's name, up to `MaxRecords` parameter groups will be listed. In either case, the parameter group's name, family, and description are listed.

### Example

The following sample code lists the parameter group *myMem14*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameterGroups
   &CacheParameterGroupName=myMem14
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action will look something like this, listing the name, family, and description for each parameter group.

```
<DescribeCacheParameterGroupsResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <DescribeCacheParameterGroupsResult>
    <CacheParameterGroups>
      <CacheParameterGroup>
        <CacheParameterGroupName>myMem14</CacheParameterGroupName>
        <CacheParameterGroupFamily>memcached1.4</CacheParameterGroupFamily>
        <Description>My custom Memcached 1.4 parameter group</Description>
      </CacheParameterGroup>
    </CacheParameterGroups>
  </DescribeCacheParameterGroupsResult>
  <ResponseMetadata>
    <RequestId>3540cc3d-af48-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParameterGroupsResponse>
```

### Example

The following sample code lists up to 10 parameter groups.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameterGroups
   &MaxRecords=10
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
```

```
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action will look something like this, listing the name, family, description and, in the case of redis5.6 if the parameter group belongs to a global datastore (isGlobal), for each parameter group.

```
<DescribeCacheParameterGroupsResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <DescribeCacheParameterGroupsResult>
    <CacheParameterGroups>
      <CacheParameterGroup>
        <CacheParameterGroupName>myRed7</CacheParameterGroupName>
        <CacheParameterGroupFamily>redis7</CacheParameterGroupFamily>
        <Description>My custom Redis OSS 7 parameter group</Description>
      </CacheParameterGroup>
      <CacheParameterGroup>
        <CacheParameterGroupName>myMem14</CacheParameterGroupName>
        <CacheParameterGroupFamily>memcached1.4</CacheParameterGroupFamily>
        <Description>My custom Memcached 1.4 parameter group</Description>
      </CacheParameterGroup>
       <CacheParameterGroup>
        <CacheParameterGroupName>myRedis56</CacheParameterGroupName>
        <CacheParameterGroupFamily>redis5.0</CacheParameterGroupFamily>
        <Description>My custom redis 5.6 parameter group</Description>
        <isGlobal>yes</isGlobal>
      </CacheParameterGroup>
    </CacheParameterGroups>
  </DescribeCacheParameterGroupsResult>
  <ResponseMetadata>
    <RequestId>3540cc3d-af48-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParameterGroupsResponse>
```

### Example

The following sample code lists the parameter group *myRed7*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameterGroups
   &CacheParameterGroupName=myRed7
   &SignatureVersion=4
```

```
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action will look something like this, listing the name, family, and description.

```
<DescribeCacheParameterGroupsResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <DescribeCacheParameterGroupsResult>
    <CacheParameterGroups>
      <CacheParameterGroup>
        <CacheParameterGroupName>myRed7</CacheParameterGroupName>
        <CacheParameterGroupFamily>redis7</CacheParameterGroupFamily>
        <Description>My custom Redis OSS 7 parameter group</Description>
      </CacheParameterGroup>
    </CacheParameterGroups>
  </DescribeCacheParameterGroupsResult>
  <ResponseMetadata>
    <RequestId>3540cc3d-af48-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParameterGroupsResponse>
```

**Example**

The following sample code lists the parameter group *myRed56*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameterGroups
   &CacheParameterGroupName=myRed56
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action will look something like this, listing the name, family, description and whether the parameter group is part of a global datastore (isGlobal).

```
<DescribeCacheParameterGroupsResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
```

```
  <DescribeCacheParameterGroupsResult>
    <CacheParameterGroups>
      <CacheParameterGroup>
        <CacheParameterGroupName>myRed56</CacheParameterGroupName>
        <CacheParameterGroupFamily>redis5.0</CacheParameterGroupFamily>
        <Description>My custom Redis 5.6 parameter group</Description>
        <isGlobal>yes</isGlobal>
      </CacheParameterGroup>
    </CacheParameterGroups>
  </DescribeCacheParameterGroupsResult>
  <ResponseMetadata>
    <RequestId>3540cc3d-af48-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParameterGroupsResponse>
```

For more information, see DescribeCacheParameterGroups.

## Listing an ElastiCache parameter group's values

You can list the parameters and their values for a parameter group using the ElastiCache console, the AWS CLI, or the ElastiCache API.

### Listing an ElastiCache parameter group's values (Console)

The following procedure shows how to list the parameters and their values for a parameter group using the ElastiCache console.

**To list a parameter group's parameters and their values using the ElastiCache console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of all available parameter groups, in the left hand navigation pane choose **Parameter Groups**.
3. Choose the parameter group for which you want to list the parameters and values by choosing the box to the left of the parameter group's name.

   The parameters and their values will be listed at the bottom of the screen. Due to the number of parameters, you may have to scroll up and down to find the parameter you're interested in.

### Listing a parameter group's values (AWS CLI)

To list a parameter group's parameters and their values using the AWS CLI, use the command `describe-cache-parameters`.

**Example**

The following sample code list all the Memcached parameters and their values for the parameter group *myMem14*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameters \
    --cache-parameter-group-name myMem14
```

For Windows:

```
aws elasticache describe-cache-parameters ^
```

```
    --cache-parameter-group-name myMem14
```

**Example**

The following sample code list all the parameters and their values for the parameter group *myRed7*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameters \
    --cache-parameter-group-name myRed7
```

For Windows:

```
aws elasticache describe-cache-parameters ^
    --cache-parameter-group-name myRed7
```

For more information, see `describe-cache-parameters`.

**Listing a parameter group's values (ElastiCache API)**

To list a parameter group's parameters and their values using the ElastiCache API, use the `DescribeCacheParameters` action.

**Example**

The following sample code list all the Memcached parameters for the parameter group *myMem14*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameters
   &CacheParameterGroupName=myMem14
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action will look something like this. This response has been truncated.

```
<DescribeCacheParametersResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <DescribeCacheParametersResult>
```

```
    <CacheClusterClassSpecificParameters>
      <CacheNodeTypeSpecificParameter>
        <DataType>integer</DataType>
        <Source>system</Source>
        <IsModifiable>false</IsModifiable>
        <Description>The maximum configurable amount of memory to use to store items,
 in megabytes.</Description>
        <CacheNodeTypeSpecificValues>
          <CacheNodeTypeSpecificValue>
            <Value>512</Value>
            <CacheClusterClass>cache.t3.micro</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>1402</Value>
            <CacheClusterClass>cache.t3.small</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>6537</Value>
            <CacheClusterClass>cache.m5.large</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>13248</Value>
            <CacheClusterClass>cache.m5.xlarge</CacheClusterClass>
          </CacheNodeTypeSpecificValue>

...output omitted...

    </CacheClusterClassSpecificParameters>
  </DescribeCacheParametersResult>
  <ResponseMetadata>
    <RequestId>6d355589-af49-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParametersResponse>
```

### Example

The following sample code list all the parameters for the parameter group *myRed7*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameters
   &CacheParameterGroupName=myRed7
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
```

```
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The response from this action will look something like this. This response has been truncated.

```
<DescribeCacheParametersResponse xmlns="http://elasticache.amazonaws.com/
doc/2013-06-15/">
  <DescribeCacheParametersResult>
    <CacheClusterClassSpecificParameters>
      <CacheNodeTypeSpecificParameter>
        <DataType>integer</DataType>
        <Source>system</Source>
        <IsModifiable>false</IsModifiable>
        <Description>The maximum configurable amount of memory to use to store items,
 in megabytes.</Description>
        <CacheNodeTypeSpecificValues>
          <CacheNodeTypeSpecificValue>
            <Value>512</Value>
            <CacheClusterClass>cache.t3.micro</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>1402</Value>
            <CacheClusterClass>cache.t3.small</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>6537</Value>
            <CacheClusterClass>cache.m5.large</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>13248</Value>
            <CacheClusterClass>cache.m5.xlarge</CacheClusterClass>
          </CacheNodeTypeSpecificValue>

...output omitted...

    </CacheClusterClassSpecificParameters>
  </DescribeCacheParametersResult>
  <ResponseMetadata>
    <RequestId>6d355589-af49-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParametersResponse>
```

For more information, see DescribeCacheParameters.

## Modifying an ElastiCache parameter group

> **Important**
>
> You cannot modify any default parameter group.

You can modify some parameter values in a parameter group. These parameter values are applied to clusters associated with the parameter group. For more information on when a parameter value change is applied to a parameter group, see Valkey and Redis OSS parameters and Memcached specific parameters.

### Modifying a parameter group (Console)

The following procedure shows how to change the `cluster-enabled` parameter's value using the ElastiCache console. You would use the same procedure to change the value of any parameter.

**To change a parameter's value using the ElastiCache console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of all available parameter groups, in the left hand navigation pane choose **Parameter Groups**.
3. Choose the parameter group you want to modify by choosing the box to the left of the parameter group's name.

   The parameter group's parameters will be listed at the bottom of the screen. You may need to page through the list to see all the parameters.
4. To modify one or more parameters, choose **Edit Parameters**.
5. In the **Edit Parameter Group:** screen, scroll using the left and right arrows until you find the `binding_protocol` parameter, then type `ascii` in the **Value** column.
6. Choose **Save Changes**.
7. For Memcached, to find the name of the parameter you changed, see Memcached specific parameters. If changes to the parameter take place *After restart*, reboot every cluster that uses this parameter group. For more information, see Rebooting clusters.

8. With Valkey and Redis OSS, to find the name of the parameter you changed, see Valkey and Redis OSS parameters. If you have a Valkey or Redis OSS (cluster mode disabled) cluster and make changes to the following parameters, you must reboot the nodes in the cluster:

   - activerehashing
   - databases

   For more information, see Rebooting nodes.

   > **Valkey or Redis OSS (Cluster Mode Enabled) parameter changes**
   >
   > If you make changes to the following parameters on a Valkey or Redis OSS (cluster mode enabled) cluster, follow the ensuing steps.
   >
   > - activerehashing
   > - databases
   >
   > 1. Create a manual backup of your cluster. See Taking manual backups.
   > 2. Delete the cluster. See Deleting clusters.
   > 3. Restore the cluster using the altered parameter group and backup to seed the new cluster. See Restoring from a backup into a new cache.
   >
   > Changes to other parameters do not require this.

**Modifying a parameter group (AWS CLI)**

To change a parameter's value using the AWS CLI, use the command `modify-cache-parameter-group`.

**Example**

With Memcached, to find the name and permitted values of the parameter you want to change, see Memcached specific parameters

The following sample code sets the value of two parameters, *chunk_size* and *chunk_size_growth_fact* on the parameter group `myMem14`.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-parameter-group \
    --cache-parameter-group-name myMem14 \
    --parameter-name-values \
        ParameterName=chunk_size,ParameterValue=96 \
        ParameterName=chunk_size_growth_fact,ParameterValue=1.5
```

For Windows:

```
aws elasticache modify-cache-parameter-group ^
    --cache-parameter-group-name myMem14 ^
    --parameter-name-values ^
        ParameterName=chunk_size,ParameterValue=96 ^
        ParameterName=chunk_size_growth_fact,ParameterValue=1.5
```

Output from this command will look something like this.

```
{
    "CacheParameterGroupName": "myMem14"
}
```

**Example**

With Valkey and Redis OSS, to find the name and permitted values of the parameter you want to change, see Valkey and Redis OSS parameters

The following sample code sets the value of two parameters, *reserved-memory-percent* and *cluster-enabled* on the parameter group `myredis7-on-30`. We set *reserved-memory-percent* to `30` (30 percent) and *cluster-enabled* to `yes` so that the parameter group can be used with Valkey or Redis OSS (cluster mode enabled) clusters (replication groups).

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-parameter-group \
    --cache-parameter-group-name myredis7-on-30 \
    --parameter-name-values \
        ParameterName=reserved-memory-percent,ParameterValue=30 \
        ParameterName=cluster-enabled,ParameterValue=yes
```

For Windows:

```
aws elasticache modify-cache-parameter-group ^
    --cache-parameter-group-name myredis7-on-30 ^
    --parameter-name-values ^
        ParameterName=reserved-memory-percent,ParameterValue=30 ^
        ParameterName=cluster-enabled,ParameterValue=yes
```

Output from this command will look something like this.

```
{
    "CacheParameterGroupName": "my-redis7-on-30"
}
```

For more information, see `modify-cache-parameter-group`.

To find the name of the parameter you changed, see Valkey and Redis OSS parameters.

If you have a Valkey or Redis OSS (cluster mode disabled) cluster and make changes to the following parameters, you must reboot the nodes in the cluster:

- activerehashing
- databases

For more information, see Rebooting nodes.

**Valkey or Redis OSS (Cluster Mode Enabled) parameter changes**

If you make changes to the following parameters on a Valkey or Redis OSS (cluster mode enabled) cluster, follow the ensuing steps.

- activerehashing
- databases

1. Create a manual backup of your cluster. See Taking manual backups.
2. Delete the cluster. See Deleting clusters.
3. Restore the cluster using the altered parameter group and backup to seed the new cluster. See Restoring from a backup into a new cache.

Changes to other parameters do not require this.

### Modifying a parameter group (ElastiCache API)

To change a parameter group's parameter values using the ElastiCache API, use the `ModifyCacheParameterGroup` action.

#### Example

With Memcached, to find the name and permitted values of the parameter you want to change, see Memcached specific parameters

The following sample code sets the value of two parameters, *chunk_size* and *chunk_size_growth_fact* on the parameter group `myMem14`.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=ModifyCacheParameterGroup
   &CacheParameterGroupName=myMem14
   &ParameterNameValues.member.1.ParameterName=chunk_size
   &ParameterNameValues.member.1.ParameterValue=96
   &ParameterNameValues.member.2.ParameterName=chunk_size_growth_fact
   &ParameterNameValues.member.2.ParameterValue=1.5
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

#### Example

With Valkey and Redis OSS, to find the name and permitted values of the parameter you want to change, see Valkey and Redis OSS parameters

The following sample code sets the value of two parameters, *reserved-memory-percent* and *cluster-enabled* on the parameter group `myredis7-on-30`. We set *reserved-memory-percent* to `30` (30 percent) and *cluster-enabled* to `yes` so that the parameter group can be used with Valkey or Redis OSS (cluster mode enabled) clusters (replication groups).

```
https://elasticache.us-west-2.amazonaws.com/
```

```
?Action=ModifyCacheParameterGroup
&CacheParameterGroupName=myredis7-on-30
&ParameterNameValues.member.1.ParameterName=reserved-memory-percent
&ParameterNameValues.member.1.ParameterValue=30
&ParameterNameValues.member.2.ParameterName=cluster-enabled
&ParameterNameValues.member.2.ParameterValue=yes
&SignatureVersion=4
&SignatureMethod=HmacSHA256
&Timestamp=20150202T192317Z
&Version=2015-02-02
&X-Amz-Credential=<credential>
```

For more information, see ModifyCacheParameterGroup.

If you have a Valkey or Redis OSS (cluster mode disabled) cluster and make changes to the following parameters, you must reboot the nodes in the cluster:

- activerehashing
- databases

For more information, see Rebooting nodes.

**Valkey or Redis OSS (Cluster Mode Enabled) parameter changes**

If you make changes to the following parameters on a Valkey or Redis OSS (cluster mode enabled) cluster, follow the ensuing steps.

- activerehashing
- databases

1. Create a manual backup of your cluster. See Taking manual backups.
2. Delete the cluster. See Deleting a cluster in ElastiCache.
3. Restore the cluster using the altered parameter group and backup to seed the new cluster. See Restoring from a backup into a new cache.

Changes to other parameters do not require this.

## Deleting an ElastiCache parameter group

You can delete a custom parameter group using the ElastiCache console, the AWS CLI, or the ElastiCache API.

You cannot delete a parameter group if it is associated with any clusters. Nor can you delete any of the default parameter groups.

### Deleting a parameter group (Console)

The following procedure shows how to delete a parameter group using the ElastiCache console.

**To delete a parameter group using the ElastiCache console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of all available parameter groups, in the left hand navigation pane choose **Parameter Groups**.
3. Choose the parameter groups you want to delete by choosing the box to the left of the parameter group's name.

   The **Delete** button will become active.
4. Choose **Delete**.

   The **Delete Parameter Groups** confirmation screen will appear.
5. To delete the parameter groups, on the **Delete Parameter Groups** confirmation screen, choose **Delete**.

   To keep the parameter groups, choose **Cancel**.

### Deleting a parameter group (AWS CLI)

To delete a parameter group using the AWS CLI, use the command `delete-cache-parameter-group`. For the parameter group to delete, the parameter group specified by `--cache-parameter-group-name` cannot have any clusters associated with it, nor can it be a default parameter group.

The following sample code deletes the *myMem14* parameter group.

**Example**

For Linux, macOS, or Unix:

```
aws elasticache delete-cache-parameter-group \
    --cache-parameter-group-name myRed7
```

For Windows:

```
aws elasticache delete-cache-parameter-group ^
    --cache-parameter-group-name myRed7
```

For more information, see delete-cache-parameter-group.

**Deleting a parameter group (ElastiCache API)**

To delete a parameter group using the ElastiCache API, use the `DeleteCacheParameterGroup` action. For the parameter group to delete, the parameter group specified by `CacheParameterGroupName` cannot have any clusters associated with it, nor can it be a default parameter group.

**Example**

With Memcached, the following sample code deletes the *myMem14* parameter group.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DeleteCacheParameterGroup
   &CacheParameterGroupName=myMem14
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

**Example**

The following sample code deletes the *myRed7* parameter group.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DeleteCacheParameterGroup
   &CacheParameterGroupName=myRed7
```

```
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &Version=2015-02-02
    &X-Amz-Credential=<credential>
```

For more information, see DeleteCacheParameterGroup.

## Engine specific parameters

### Valkey and Redis OSS

Most Valkey 8 parameters are compatible with Redis OSS 7.1 parameters. Valkey 7.2 parameters are the same as Redis OSS 7 parameters.

If you do not specify a parameter group for your Valkey or Redis OSS cluster, then a default parameter group appropriate to your engine version will be used. You can't change the values of any parameters in the default parameter group. However, you can create a custom parameter group and assign it to your cluster at any time as long as the values of conditionally modifiable parameters are the same in both parameter groups. For more information, see Creating an ElastiCache parameter group.

### Topics


- Valkey and Redis OSS parameters
- Memcached specific parameters

## Valkey and Redis OSS parameters

### Topics


- Valkey 9.1 and Valkey 9.0 parameter changes
- Valkey 8.2 parameter changes
- Valkey 8.1 parameter changes
- Valkey 8.0 parameter changes
- Valkey 7.2 and Redis OSS 7 parameter changes
- Redis OSS 6.x parameter changes
- Redis OSS 5.0.3 parameter changes
- Redis OSS 5.0.0 parameter changes
- Redis OSS 4.0.10 parameter changes
- Redis OSS 3.2.10 parameter changes
- Redis OSS 3.2.6 parameter changes
- Redis OSS 3.2.4 parameter changes
- Redis OSS 2.8.24 (enhanced) added parameters
- Redis OSS 2.8.23 (enhanced) added parameters
- Redis OSS 2.8.22 (enhanced) added parameters
- Redis OSS 2.8.21 added parameters
- Redis OSS 2.8.19 added parameters
- Redis OSS 2.8.6 added parameters
- Redis OSS 2.6.13 parameters
- Valkey node-type specific parameters
- Redis OSS node-type specific parameters


### Valkey 9.1 and Valkey 9.0 parameter changes

**Parameter group family:** valkey9

> **Note**
>
> - Valkey 9.0 and above parameter groups are incompatible with Valkey 8.x and Redis OSS.

New parameter group families for Valkey 9.1 and Valkey 9.0:

- `default.valkey9` – Default parameter group for Valkey 9.1 and Valkey 9.0 (cluster mode disabled).
- `default.valkey9.cluster.on` – Default parameter group for Valkey 9.1 and Valkey 9.0 with cluster mode enabled.

### New customer-visible parameter in Valkey 9.0

| Parameter | Description | Values | Default | Modifiable |
|---|---|---|---|---|
| `cluster-databases` | Number of databases available in cluster mode. Set at cluster creation time. | 1 to 10,000 | 0 | No (set at creation) |

### Valkey 8.2 parameter changes

**Parameter group family:** valkey8

> **Note**
>
> - Valkey 8.2 parameter changes don't apply to Valkey 8.1
> - Valkey 8.0 and above parameter groups are incompatible with Redis OSS 7.2.4.
> - in Valkey 8.2, the following commands are unavailable for serverless caches: `commandlog`, `commandlog get`, `commandlog help`, `commandlog len`, and `commandlog reset`.

### New parameter groups in Valkey 8.2

| Name | Details | Description |
|---|---|---|
| search-fanout-target-mode (added in 8.2) | Default: client<br><br>Type: string | The search-fanout-target-mode configuration parameter controls how search queries are distribut |

| Name | Details | Description |
| --- | --- | --- |
| | Modifiable: Yes<br><br>Changes Take Effect: Immediately | ed across nodes in a Valkey cluster environment. This setting accepts two values: "throughput" which optimizes for maximum throughput by randomly distributing search queries across all cluster nodes regardless of client type or READONLY status, and "client" which respects client connection character istics by routing non-READO NLY clients to primary nodes only, READONLY clients on replica connections to replica nodes only, and READONLY clients on primary connectio ns randomly across all nodes.<br><br>The default behavior is "client' mode, meaning the system will respect client connection types and READONLY status for query routing decisions. Use throughput mode for high-volume search workloads where maximum cluster resource utilization is desired, and client mode when you want to maintain read/writ e separation and respect application-level READONLY connection patterns. |

| Name | Details | Description |
| --- | --- | --- |
| search-default-timeout-ms | Default: 50000<br><br>Permitted values: 1 to 60000<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | The default Valkey search query timeout (in milliseconds). |
| search-enable-partial-results | Default: yes<br><br>Permitted values: yes, no<br><br>Type: boolean<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Configures the query failure behavior for Valkey search. When enabled, search queries will return partial results if timeouts occur on one or more shards. When disabled, any shard timeout will cause the entire search query to fail and return an error. |

### Valkey 8.1 parameter changes

**Parameter group family:** valkey8

> **Note**
>
> - Valkey 8.1 parameter changes don't apply to Valkey 8.0

- Valkey 8.0 and above parameter groups are incompatible with Redis OSS 7.2.4.
- in Valkey 8.1, the following commands are unavailable for serverless caches: `commandlog`, `commandlog get`, `commandlog help`, `commandlog len`, and `commandlog reset`.

**New parameter groups in Valkey 8.1**

| Name | Details | Description |
|---|---|---|
| commandlog-request-larger-than (added in 8.1) | Default: 1048576<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | The maximum size, in bytes, for requests to be logged by the Valkey Command Log feature. |
| commandlog-large-request-max-len (added in 8.1) | Default: 128<br><br>Permitted values: 0-1024<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | The maximum length of the Valkey Command Log for requests. |
| commandlog-reply-larger-than (added in 8.1) | Default: 1048576<br><br>Type: integer<br><br>Modifiable: Yes | The maximum size, in bytes, for responses to be logged by the Valkey Command Log feature. |

| Name | Details | Description |
| --- | --- | --- |
| | Changes Take Effect: Immediately | |
| commandlog-large-reply-max-len (added in 8.1) | Default: 128<br>Permitted values: 0-1024<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | The maximum length of the Valkey Command Log for responses. |

### Valkey 8.0 parameter changes

**Parameter group family:** valkey8

> **Note**
>
> Redis OSS 7.2.4 is incompatible with Valkey 8 and above parameter groups.

### Specific parameter changes in Valkey 8.0

| Name | Details | Description |
| --- | --- | --- |
| repl-backlog-size | Default: 10485760<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | The size, in bytes, of the primary node backlog buffer. The backlog is used for recording updates to data at the primary node. When a read replica connects to the primary, it attempts to perform a partial sync (psync), where it applies data |

| Name | Details | Description |
| --- | --- | --- |
| | | from the backlog to catch up with the primary node. If the psync fails, then a full sync is required.<br><br>The minimum value for this parameter is 16384.<br><br>Note: Beginning with Redis OSS 2.8.22, this parameter applies to the primary cluster as well as the read replicas. |
| maxmemory-samples | Default: 3<br><br>Permitted values: 1 to 64<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | For least-recently-used (LRU) and time-to-live (TTL) calculations, this parameter represents the sample size of keys to check. By default, Redis OSS chooses 3 keys and uses the one that was used least recently. |

**New parameter groups in Valkey 8.0**

| Name | Details | Description |
| --- | --- | --- |
| extended-redis-compatibility | Permitted values: yes, no<br><br>Default: yes<br><br>Type: boolean<br><br>Modifiable: Yes | Extended Redis OSS compatibility mode makes Valkey pretend to be Redis OSS 7.2. Enable this only if you have problems with tools or clients.<br><br>Customer-facing impacts: |

| Name | Details | Description |
| --- | --- | --- |
| | Changes take place: immediately | • LOADING - Redis OSS is loading the dataset in memory<br>• BUSY - Redis OSS is busy<br>• MISCONF - Redis OSS is configured in either of these ways:<br>  • The HELLO command returns "server" => "redis" and "version" => "7.2.4" (our Redis OSS compatibility version).<br>  • The INFO field for mode is called "redis_mode". |

**Removed parameter groups in Valkey 8.0**

| Name | Details | Description |
| --- | --- | --- |
| lazyfree-lazy-eviction | Permitted values: yes, no<br>Default: no<br>Type: boolean<br>Modifiable: Yes<br>Changes take place: immediately | Performs an asynchronous delete on evictions. |
| lazyfree-lazy-expire | Permitted values: yes, no<br>Default: no | Performs an asynchronous delete on expired keys. |

| Name | Details | Description |
| --- | --- | --- |
| | Type: boolean<br><br>Modifiable: Yes<br><br>Changes take place: immediately | |
| lazyfree-lazy-server-del | Permitted values: yes, no<br><br>Default: no<br><br>Type: boolean<br><br>Modifiable: Yes<br><br>Changes take place: immediately | Performs an asynchronous delete for commands which update values. |
| lazyfree-lazy-user-del | Default: no<br><br>Type: string<br><br>Modifiable: Yes<br><br>Changes take effect: Immediately across all nodes in the cluster | When the value is set to yes, the DEL command acts the same as UNLINK. |
| replica-lazy-flush | Default: yes<br><br>Type: boolean<br><br>Modifiable: No<br><br>Former name: slave-lazy-flush | Performs an asynchronous flushDB during replica sync. |

### Valkey 7.2 and Redis OSS 7 parameter changes

**Parameter group family:** valkey7

Valkey 7.2 default parameter groups are as follows:

- `default.valkey7` – Use this parameter group, or one derived from it, for Valkey (cluster mode disabled) clusters and replication groups.
- `default.valkey7.cluster.on` – Use this parameter group, or one derived from it, for Valkey (cluster mode enabled) clusters and replication groups.

**Parameter group family:** redis7

Redis OSS 7 default parameter groups are as follows:

- `default.redis7` – Use this parameter group, or one derived from it, for Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis7.cluster.on` – Use this parameter group, or one derived from it, for Redis OSS (cluster mode enabled) clusters and replication groups.

### Specific parameter changes

Parameters added in Redis OSS 7 are as follows. Valkey 7.2 also supports these parameters.

| Name | Details | Description |
| --- | --- | --- |
| `cluster-allow-pubsubshard-when-down` | Permitted values: `yes`, `no`<br>Default: `yes`<br>Type: string<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | When set to the default of yes, allows nodes to serve pubsub shard traffic while the cluster is in a down state, as long as it believes it owns the slots. |

| Name | Details | Description |
| --- | --- | --- |
| `cluster-preferred-endpoint-type` | Permitted values: `ip`, `tls-dynamic`<br>Default: `tls-dynamic`<br>Type: string<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | This value controls what endpoint is returned for MOVED/ASKING requests as well as the endpoint field for `CLUSTER SLOTS` and `CLUSTER SHARDS`. When the value is set to ip, the node will advertise its ip address. When the value is set to tls-dynamic, the node will advertise a hostname when encryption-in-transit is enabled and an ip address otherwise. |
| `latency-tracking` | Permitted values: `yes`, `no`<br>Default: no<br>Type: string<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | When set to yes tracks the per command latencies and enables exporting the percentile distribution via the `INFO` latency statistics command, and cumulative latency distributions (histograms) via the `LATENCY` command. |
| `hash-max-listpack-entries` | Permitted values: `0+`<br>Default: `512`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | The maximum number of hash entries in order for the dataset to be compressed. |

| Name | Details | Description |
| --- | --- | --- |
| `hash-max-listpack-value` | Permitted values: `0+`<br>Default: `64`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | The threshold of biggest hash entries in order for the dataset to be compressed. |
| `zset-max-listpack-entries` | Permitted values: `0+`<br>Default: `128`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | The maximum number of sorted set entries in order for the dataset to be compressed. |
| `zset-max-listpack-value` | Permitted values: `0+`<br>Default: `64`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | The threshold of biggest sorted set entries in order for the dataset to be compressed. |

Parameters changed in Redis OSS 7 are as follows.

| Name | Details | Description |
| --- | --- | --- |
| activerehashing | Modifiable: no. In Redis OSS 7, this parameter is hidden and enabled by default. In order to disable it, you need to create a support case. | Modifiable was yes. |

Parameters removed in Redis OSS 7 are as follows.

| Name | Details | Description |
| --- | --- | --- |
| hash-max-ziplist-entries | Permitted values: 0+<br>Default: 512<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | Use listpack instead of ziplist for representing small hash encoding |
| hash-max-ziplist-value | Permitted values: 0+<br>Default: 64<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | Use listpack instead of ziplist for representing small hash encoding |

| Name | Details | Description |
| --- | --- | --- |
| `zset-max-ziplist-entries` | Permitted values: `0+`<br>Default: `128`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | Use `listpack` instead of `ziplist` for representing small hash encoding. |
| `zset-max-ziplist-value` | Permitted values: `0+`<br>Default: `64`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | Use `listpack` instead of `ziplist` for representing small hash encoding. |
| `list-max-ziplist-size` | Permitted values:<br>Default: `-2`<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster. | The number of entries allowed per internal list node. |

### Redis OSS 6.x parameter changes

**Parameter group family:** redis6.x

Redis OSS 6.x default parameter groups are as follows:

- `default.redis6.x` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis6.x.cluster.on` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode enabled) clusters and replication groups.

> **Note**
>
> In Redis OSS engine version 6.2, when the r6gd node family was introduced for use with Data tiering in ElastiCache, only *noeviction*, *volatile-lru* and *allkeys-lru* max-memory policies are supported with r6gd node types.

For more information, see ElastiCache version 6.2 for Redis OSS (enhanced) and ElastiCache version 6.0 for Redis OSS (enhanced).

Parameters added in Redis OSS 6.x are as follows.

| Details | Description | |
|---|---|---|
| `acl-pubsub-default (added in 6.2)` | Permitted values: `resetchannels , allchannels`<br><br>Default: `allchannels`<br><br>Type: string<br><br>Modifiable: Yes<br><br>Changes take effect: The existing Redis OSS users associated to the cluster will continue to have existing | Default pubsub channel permissions for ACL users deployed to this cluster. |

| Details | Description | |
|---|---|---|
| | permissions. Either update the users or reboot the cluster to update the existing Redis OSS users. | |
| `cluster-allow-reads-when-down (added in 6.0)` | Default: no<br><br>Type: string<br><br>Modifiable: Yes<br><br>Changes take effect: Immediately across all nodes in the cluster | When set to yes, a Redis OSS (cluster mode enabled) replication group continues to process read commands even when a node is not able to reach a quorum of primaries.<br><br>When set to the default of no, the replication group rejects all commands. We recommend setting this value to yes if you are using a cluster with fewer than three node groups or your application can safely handle stale reads. |
| `tracking-table-max-keys (added in 6.0)` | Default: 1,000,000<br><br>Type: number<br><br>Modifiable: Yes<br><br>Changes take effect: Immediately across all nodes in the cluster | To assist client-side caching, Redis OSS supports tracking which clients have accessed which keys.<br><br>When the tracked key is modified, invalidation messages are sent to all clients to notify them their cached values are no longer valid. This value enables you to specify the upper bound of this table. After this parameter value is exceeded, clients are sent invalidation randomly. This value should be tuned to limit memory usage while still keeping track of enough keys. Keys are also invalidated under low memory conditions. |

| Details | Description | |
|---|---|---|
| `acllog-max-len (added in 6.0)` | Default: 128<br>Type: number<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster | This value corresponds to the max number of entries in the ACL log. |
| `active-expire-effort (added in 6.0)` | Default: 1<br>Type: number<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster | Redis OSS deletes keys that have exceeded their time to live by two mechanisms. In one, a key is accessed and is found to be expired. In the other, a periodic job samples keys and causes those that have exceeded their time to live to expire. This parameter defines the amount of effort that Redis OSS uses to expire items in the periodic job.<br>The default value of 1 tries to avoid having more than 10 percent of expired keys still in memory. It also tries to avoid consuming more than 25 percent of total memory and to add latency to the system. You can increase this value up to 10 to increase the amount of effort spent on expiring keys. The tradeoff is higher CPU and potentially higher latency. We recommend a value of 1 unless you are seeing high memory usage and can tolerate an increase in CPU utilization. |

| Details | Description | |
|---|---|---|
| `lazyfree-lazy-user-del (added in 6.0)` | Default: no<br>Type: string<br>Modifiable: Yes<br>Changes take effect: Immediately across all nodes in the cluster | When the value is set to yes, the `DEL` command acts the same as `UNLINK`. |

Parameters removed in Redis OSS 6.x are as follows.

| Name | Details | Description |
|---|---|---|
| `lua-replicate-commands` | Permitted values: yes/no<br>Default: yes<br>Type: boolean<br>Modifiable: Yes<br>Changes take effect: Immediately | Always enable Lua effect replication or not in Lua scripts |

The following table lists the parameters changed in Redis OSS 6.x.

| Name | Details | Description |
|---|---|---|
| `databases` | Permitted values: 1 to 10000<br>Default: 16<br>Type: integer | Redis OSS 6.0 decreased the maximum number of allowed databases from 1200000 to 10000.<br>When cluster mode is enabled, only database 0 is available regardless of this parameter's |

| Name | Details | Description |
| --- | --- | --- |
| | Modifiable: Yes<br><br>Changes take effect: After Restart | value. For multiple databases in cluster mode, see the `cluster-databases` parameter in Valkey 9.0 and later. |

**Redis OSS 5.0.3 parameter changes**

**Parameter group family:** redis5.0

Redis OSS 5.0 default parameter groups

- `default.redis5.0` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis5.0.cluster.on` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode enabled) clusters and replication groups.

**Parameters added in Redis OSS 5.0.3**

| Name | Details | Description |
| --- | --- | --- |
| `rename-commands` | Default: none<br><br>Type: string<br><br>Modifiable: Yes<br><br>Changes take effect: Immediately across all nodes in the cluster | A space-separated list of renamed Redis OSS commands. The following is a restricted list of commands available for renaming:<br><br>`APPEND AUTH BITCOUNT BITFIELD BITOP BITPOS BLPOP BRPOP BRPOPLPUSH BZPOPMIN BZPOPMAX CLIENT CLUSTER COMMAND DBSIZE DECR DECRBY DEL DISCARD DUMP ECHO EVAL EVALSHA EXEC EXISTS EXPIRE EXPIREAT FLUSHALL FLUSHDB GEOADD GEOHASH GEOPOS GEODIST GEORADIUS GEORADIUSBYMEMBER GET GETBIT GETRANGE GETSET HDEL HEXISTS HGET HGETALL HINCRBY HINCRBYFLOAT HKEYS HLEN HMGET HMSET HSET` |

| Name | Details | Description |
| --- | --- | --- |
| | | HSETNX HSTRLEN HVALS INCR INCRBY INCRBYFLOAT INFO KEYS LASTSAVE LINDEX LINSERT LLEN LPOP LPUSH LPUSHX LRANGE LREM LSET LTRIM MEMORY MGET MONITOR MOVE MSET MSETNX MULTI OBJECT PERSIST PEXPIRE PEXPIREAT PFADD PFCOUNT PFMERGE PING PSETEX PSUBSCRIBE PUBSUB PTTL PUBLISH PUNSUBSCRIBE RANDOMKEY READONLY READWRITE RENAME RENAMENX RESTORE ROLE RPOP RPOPLPUSH RPUSH RPUSHX SADD SCARD SCRIPT SDIFF SDIFFSTORE SELECT SET SETBIT SETEX SETNX SETRANGE SINTER SINTERSTORE SISMEMBER SLOWLOG SMEMBERS SMOVE SORT SPOP SRANDMEMBER SREM STRLEN SUBSCRIBE SUNION SUNIONSTORE SWAPDB TIME TOUCH TTL TYPE UNSUBSCRIBE UNLINK UNWATCH WAIT WATCH ZADD ZCARD ZCOUNT ZINCRBY ZINTERSTORE ZLEXCOUNT ZPOPMAX ZPOPMIN ZRANGE ZRANGEBYLEX ZREVRANGEBYLEX ZRANGEBYSCORE ZRANK ZREM ZREMRANGEBYLEX ZREMRANGEBYRANK ZREMRANGEBYSCORE ZREVRANGE ZREVRANGEBYSCORE ZREVRANK ZSCORE ZUNIONSTORE SCAN SSCAN HSCAN ZSCAN XINFO XADD XTRIM XDEL XRANGE XREVRANGE XLEN XREAD XGROUP XREADGROUP XACK XCLAIM XPENDING GEORADIUS_RO GEORADIUSBYMEMBER_RO LOLWUT XSETID SUBSTR |

For more information, see ElastiCache version 5.0.6 for Redis OSS (enhanced).

**Redis OSS 5.0.0 parameter changes**

**Parameter group family:** redis5.0

Redis OSS 5.0 default parameter groups

- `default.redis5.0` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis5.0.cluster.on` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode enabled) clusters and replication groups.

**Parameters added in Redis OSS 5.0**

| Name | Details | Description |
|---|---|---|
| `stream-node-max-bytes` | Permitted values: 0+<br>Default: 4096<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately | The stream data structure is a radix tree of nodes that encode multiple items inside. Use this configuration to specify the maximum size of a single node in radix tree in Bytes. If set to 0, the size of the tree node is unlimited. |
| `stream-node-max-entries` | Permitted values: 0+<br>Default: 100<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately | The stream data structure is a radix tree of nodes that encode multiple items inside. Use this configuration to specify the maximum number of items a single node can contain before switching to a new node when appending new stream entries. If set to 0, the number of items in the tree node is unlimited |

| Name | Details | Description |
| --- | --- | --- |
| active-defrag-max-scan-fields | Permitted values: 1 to 1000000<br>Default: 1000<br>Type: integer<br>Modifiable: Yes<br>Changes take effect: Immediately | Maximum number of set/hash/zset/list fields that will be processed from the main dictionary scan |
| lua-replicate-commands | Permitted values: yes/no<br>Default: yes<br>Type: boolean<br>Modifiable: Yes<br>Changes take effect: Immediately | Always enable Lua effect replication or not in Lua scripts |
| replica-ignore-maxmemory | Default: yes<br>Type: boolean<br>Modifiable: No | Determines if replica ignores `maxmemory` setting by not evicting items independent from the primary |

**Parameters changed in Redis OSS 5.0**

| Name | Details | Description |
| --- | --- | --- |
| active-defrag- | Permitted values: 1-75<br>Default: 5 | Minimal effort for defrag in CPU percentage. |

| Name | Details | Description |
| --- | --- | --- |
| `cycle-min` | Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | |

Redis OSS has renamed several parameters in engine version 5.0 in response to community feedback. For more information, see What's New in Redis OSS 5?. The following table lists the new names and how they map to previous versions.

**Parameters renamed in Redis OSS 5.0**

| Name | Details | Description |
| --- | --- | --- |
| `replica-lazy-flush` | Default: yes<br>Type: boolean<br>Modifiable: No<br>Former name: slave-lazy-flush | Performs an asynchronous flushDB during replica sync. |
| `client-output-buffer-limit-replica-hard-limit` | Default: For values see Redis OSS node-type specific parameters<br>Type: integer<br>Modifiable: No<br>Former name: client-output-buffer-limit-slave-hard-limit | For Redis OSS read replicas: If a client's output buffer reaches the specified number of bytes, the client will be disconnected. |
| `client-output-buffer-limit-replica-s` | Default: For values see Redis OSS node-type specific parameters<br>Type: integer | For Redis OSS read replicas: If a client's output buffer reaches the specified number of bytes, the client will be disconnected, but only if this condition persists for `client-output-` |

| Name | Details | Description |
| --- | --- | --- |
| `oft-limit` | Modifiable: No<br><br>Former name: client-output-buffer-limit-slave-soft-limit | `buffer-limit-replica-soft-seconds` . |
| `client-output-buffer-limit-replica-soft-seconds` | Default: 60<br><br>Type: integer<br><br>Modifiable: No<br><br>Former name: client-output-buffer-limit-slave-soft-seconds | For Redis OSS read replicas: If a client's output buffer remains at `client-output-buffer-limit-replica-soft-limit` bytes for longer than this number of seconds, the client will be disconnected. |
| `replica-allow-chaining` | Default: no<br><br>Type: string<br><br>Modifiable: No<br><br>Former name: slave-allow-chaining | Determines whether a read replica in Redis OSS can have read replicas of its own. |
| `min-replicas-to-write` | Default: 0<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Former name: min-slaves-to-write<br><br>Changes Take Effect: Immediately | The minimum number of read replicas which must be available in order for the primary node to accept writes from clients. If the number of available replicas falls below this number, then the primary node will no longer accept write requests.<br><br>If either this parameter or min-replicas-max-lag is 0, then the primary node will always accept writes requests, even if no replicas are available. |

| Name | Details | Description |
| --- | --- | --- |
| `min-replicas-max-lag` | Default: 10<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Former name: min-slaves-max-lag<br><br>Changes Take Effect: Immediately | The number of seconds within which the primary node must receive a ping request from a read replica. If this amount of time passes and the primary does not receive a ping, then the replica is no longer considered available. If the number of available replicas drops below min-replicas-to-write, then the primary will stop accepting writes at that point.<br><br>If either this parameter or min-replicas-to-write is 0, then the primary node will always accept write requests, even if no replicas are available. |
| `close-on-replica-write` | Default: yes<br><br>Type: boolean<br><br>Modifiable: Yes<br><br>Former name: close-on-slave-write<br><br>Changes Take Effect: Immediately | If enabled, clients who attempt to write to a read-only replica will be disconnected. |

**Parameters removed in Redis OSS 5.0**

| Name | Details | Description |
| --- | --- | --- |
| `repl-timeout` | Default: 60<br><br>Modifiable: No | Parameter is not available in this version. |

### Redis OSS 4.0.10 parameter changes

**Parameter group family:** redis4.0

Redis OSS 4.0.x default parameter groups

- `default.redis4.0` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis4.0.cluster.on` – Use this parameter group, or one derived from it, for Valkey or Redis OSS (cluster mode enabled) clusters and replication groups.

### Parameters changed in Redis OSS 4.0.10

| Name | Details | Description |
| --- | --- | --- |
| `maxmemory-policy` | Permitted values: `allkeys-lru` , `volatile-lru` , **`allkeys-lfu`** , **`volatile-lfu`** , `allkeys-random` , `volatile-random` , `volatile-ttl` , `noeviction`<br><br>Default: volatile-lru<br><br>Type: string<br><br>Modifiable: Yes<br><br>Changes take place: immediately | `maxmemory-policy` was added in version 2.6.13. In version 4.0.10 two new permitted values are added: `allkeys-lfu` , which will evict any key using approximated LFU, and `volatile-lfu` , which will evict usin g approximated LFU among the keys with an expire set. In version 6.2, when the r6gd node family was introduced for use with data-tiering, only `noeviction` , `volatile-lru` and `allkeys-lru` max-memory policies are supported with r6gd node types. |

### Parameters added in Redis OSS 4.0.10

| Name | Details | Description |
| --- | --- | --- |
| **Async deletion parameters** | | |

| Name | Details | Description |
| --- | --- | --- |
| `lazyfree-lazy-eviction` | Permitted values: yes/no<br>Default: no<br>Type: boolean<br>Modifiable: Yes<br>Changes take place: immediately | Performs an asynchronous delete on evictions. |
| `lazyfree-lazy-expire` | Permitted values: yes/no<br>Default: no<br>Type: boolean<br>Modifiable: Yes<br>Changes take place: immediately | Performs an asynchronous delete on expired keys. |
| `lazyfree-lazy-server-del` | Permitted values: yes/no<br>Default: no<br>Type: boolean<br>Modifiable: Yes<br>Changes take place: immediately | Performs an asynchronous delete for commands which update values. |

| Name | Details | Description |
| --- | --- | --- |
| `slave-lazy-flush` | Permitted values: N/A<br>Default: no<br>Type: boolean<br>Modifiable: No<br>Changes take place: N/A | Performs an asynchronous flushDB during slave sync. |
| **LFU parameters** | | |
| `lfu-log-factor` | Permitted values: any integer > 0<br>Default: 10<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Set the log factor, which determines the number of key hits to saturate the key counter. |
| `lfu-decay-time` | Permitted values: any integer<br>Default: 1<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | The amount of time in minutes to decrement the key counter. |
| **Active defragmentation parameters** | | |

| Name | Details | Description |
| --- | --- | --- |
| `activedefrag` | Permitted values: yes/no<br>Default: no<br>Type: boolean<br>Modifiable: Yes<br>Changes take place: immediately | Enables active defragmentation.<br>**Note**<br>In Valkey and Redis OSS versions 7.0 and above, AWS may automatically perform defragmentation when operationally necessary, regardless of this setting. |
| `active-defrag-ignore-bytes` | Permitted values: 10485760-104857600<br>Default: 104857600<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Minimum amount of fragmentation waste to start active defrag. |

| Name | Details | Description |
| --- | --- | --- |
| `active-defrag-threshold-lower` | Permitted values: 1-100<br>Default: 10<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Minimum percentage of fragmentation to start active defrag. |
| `active-defrag-threshold-upper` | Permitted values: 1-100<br>Default: 100<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Maximum percentage of fragmentation at which we use maximum effort. |
| `active-defrag-cycle-min` | Permitted values: 1-75<br>Default: 25<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Minimal effort for defrag in CPU percentage. |

| Name | Details | Description |
| --- | --- | --- |
| `active-defrag-cycle-max` | Permitted values: 1-75<br>Default: 75<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Maximal effort for defrag in CPU percentage. |
| **Client output buffer parameters** | | |
| `client-query-buffer-limit` | Permitted values: 1048576-1073741824<br>Default: 1073741824<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Max size of a single client query buffer. |
| `proto-max-bulk-len` | Permitted values: 1048576-536870912<br>Default: 536870912<br>Type: integer<br>Modifiable: Yes<br>Changes take place: immediately | Max size of a single element request. |

### Redis OSS 3.2.10 parameter changes

**Parameter group family:** redis3.2

ElastiCache for Redis OSS 3.2.10 there are no additional parameters supported.

### Redis OSS 3.2.6 parameter changes

**Parameter group family:** redis3.2

For Redis OSS 3.2.6 there are no additional parameters supported.

### Redis OSS 3.2.4 parameter changes

**Parameter group family:** redis3.2

Beginning with Redis OSS 3.2.4 there are two default parameter groups.

- `default.redis3.2` – When running Redis OSS 3.2.4, specify this parameter group or one derived from it, if you want to create a Valkey or Redis OSS (cluster mode disabled) replication group and still use the additional features of Redis OSS 3.2.4.
- `default.redis3.2.cluster.on` – Specify this parameter group or one derived from it, when you want to create a Valkey or Redis OSS (cluster mode enabled) replication group.

**Topics**


- New parameters for Redis OSS 3.2.4
- Parameters changed in Redis OSS 3.2.4 (enhanced)


#### New parameters for Redis OSS 3.2.4

**Parameter group family:** redis3.2

For Redis OSS 3.2.4 the following additional parameters are supported.

| Name | Details | Description |
| --- | --- | --- |
| `list-max-ziplist-size` | Default: -2<br><br>Type: integer | Lists are encoded in a special way to save space. The number of entries allowed per internal list node can be specified as a fixed maximum size or a maximum number of |

| Name | Details | Description |
| --- | --- | --- |
| | Modifiable: No | elements. For a fixed maximum size, use -5 through -1, meaning:<br>• -5: max size: 64 Kb - not recommended for normal workloads<br>• -4: max size: 32 Kb - not recommended<br>• -3: max size: 16 Kb - not recommended<br>• -2: max size: 8 Kb - recommended<br>• -1: max size: 4 Kb - recommended<br>• Positive numbers mean store up to exactly that number of elements per list node. |

| Name | Details | Description |
| --- | --- | --- |
| `list-compress-depth` | Default: 0<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Lists may also be compressed. Compress depth is the number of quicklist ziplist nodes from each side of the list to exclude from compression. The head and tail of the list are always uncompressed for fast push and pop operations. Settings are:<br><br>• 0: Disable all compression.<br>• 1: Start compressing with the 1st node in from the head and tail.<br><br>[head]->node->node->...->node->[tail]<br><br>All nodes except [head] and [tail] compress.<br>• 2: Start compressing with the 2nd node in from the head and tail.<br><br>[head]->[next]->node->node->...->node->[prev]->[tail]<br><br>[head], [next], [prev], [tail] do not compress. All other nodes compress.<br>• Etc. |

| Name | Details | Description |
| --- | --- | --- |
| `cluster-enabled` | Default: no/yes *<br><br>Type: string<br><br>Modifiable: No | Indicates whether this is a Valkey or Redis OSS (cluster mode enabled) replication group in cluster mode (yes) or a Valkey or Redis OSS (cluster mode enabled) replication group in non-cluster mode (no). Valkey or Redis OSS (cluster mode enabled) replication groups in cluster mode can partition their data across up to 500 node groups.<br><br>* Redis OSS 3.2.*x* has two default parameter groups.<br><br>• `default.redis3.2` – default value no.<br>• `default.redis3.2.cluster.on` – default value `yes`.<br><br>. |
| `cluster-require-full-coverage` | Default: no<br><br>Type: boolean<br><br>Modifiable: yes<br><br>Changes Take Effect: Immediately | When set to `yes`, Valkey or Redis OSS (cluster mode enabled) nodes in cluster mode stop accepting queries if they detect there is at least one hash slot uncovered (no available node is serving it). This way if the cluster is partially down, the cluster becomes unavailable. It automatically becomes available again as soon as all the slots are covered again.<br><br>However, sometimes you want the subset of the cluster which is working to continue to accept queries for the part of the key space that is still covered. To do so, just set the `cluster-require-full-coverage` option to no. |

| Name | Details | Description |
| --- | --- | --- |
| `hll-sparse-max-bytes` | Default: 3000<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | HyperLogLog sparse representation bytes limit. The limit includes the 16 byte header. When a HyperLogLog using the sparse representation crosses this limit, it is converted into the dense representation.<br><br>A value greater than 16000 is not recommended, because at that point the dense representation is more memory efficient.<br><br>We recommend a value of about 3000 to have the benefits of the space-efficient encoding without slowing down PFADD too much, which is O(N) with the sparse encoding. The value can be raised to ~10000 when CPU is not a concern, but space is, and the data set is composed of many HyperLogLogs with cardinality in the 0 - 15000 range. |

| Name | Details | Description |
| --- | --- | --- |
| `reserved-memory-percent` | Default: 25<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | The percent of a node's memory reserved for nondata use. By default, the Redis OSS data footprint grows until it consumes all of the node's memory. If this occurs, then node performance will likely suffer due to excessive memory paging. By reserving memory, you can set aside some of the available memory for non-Redis OSS purposes to help reduce the amount of paging.<br><br>This parameter is specific to ElastiCache, and is not part of the standard Redis OSS distribution.<br><br>For more information, see `reserved-memory` and Managing reserved memory for Valkey and Redis OSS. |

**Parameters changed in Redis OSS 3.2.4 (enhanced)**

**Parameter group family:** redis3.2

For Redis OSS 3.2.4 the following parameters were changed.

| Name | Details | Change |
| --- | --- | --- |
| `activerehashing` | Modifiable: Yes if the parameter group is not associated with any clusters. Otherwise, no. | Modifiable was No. |
| `databases` | Modifiable: Yes if the parameter group is not associated with any clusters. Otherwise, no. | Modifiable was No. |

| Name | Details | Change |
| --- | --- | --- |
| appendonly | Default: off<br>Modifiable: No | If you want to upgrade from an earlier Redis OSS version, you must first turn appendonly off. |
| appendfsync | Default: off<br>Modifiable: No | If you want to upgrade from an earlier Redis OSS version, you must first turn appendfsync off. |
| repl-timeout | Default: 60<br>Modifiable: No | Is now unmodifiable with a default of 60. |
| tcp-keepalive | Default: 300 | Default was 0. |
| list-max-ziplist-entries | | Parameter is no longer available. |
| list-max-ziplist-value | | Parameter is no longer available. |

**Redis OSS 2.8.24 (enhanced) added parameters**

**Parameter group family:** redis2.8

For Redis OSS 2.8.24 there are no additional parameters supported.

**Redis OSS 2.8.23 (enhanced) added parameters**

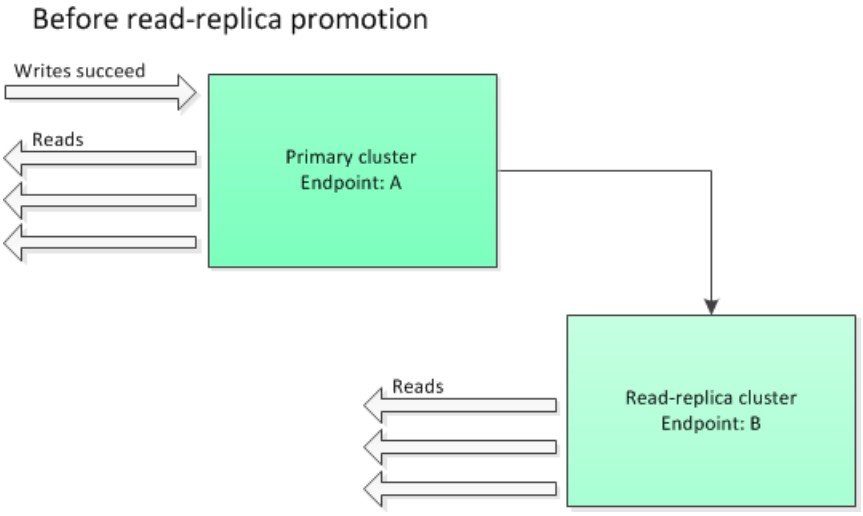
**Parameter group family:** redis2.8

For Redis OSS 2.8.23 the following additional parameter is supported.

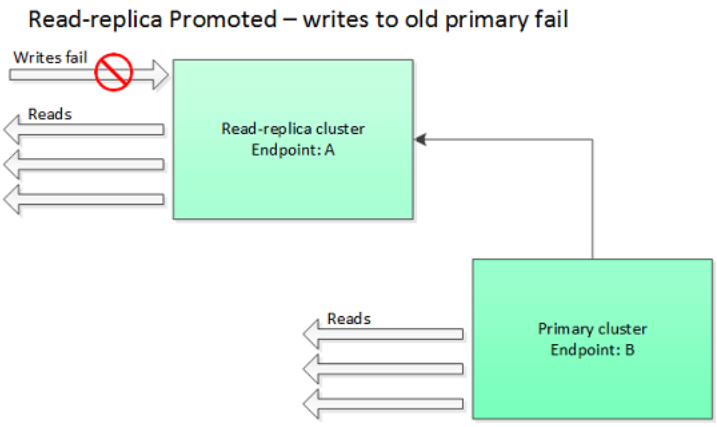
| Name | Details | Description |
| --- | --- | --- |
| `close-on-slave-write` | Default: yes<br><br>Type: string (yes/no)<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | If enabled, clients who attempt to write to a read-only replica will be disconnected. |

**How close-on-slave-write works**

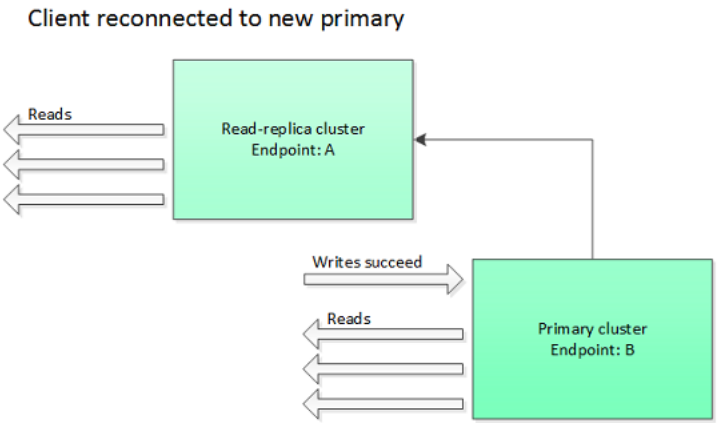
The `close-on-slave-write` parameter is introduced by Amazon ElastiCache to give you more control over how your cluster responds when a primary node and a read replica node swap roles due to promoting a read replica to primary.




If the read-replica cluster is promoted to primary for any reason other than a Multi-AZ enabled replication group failing over, the client will continue trying to write to endpoint A. Because endpoint A is now the endpoint for a read-replica, these writes will fail. This is the behavior for Redis OSS before ElastiCache introducing `close-on-replica-write` and the behavior if you disable `close-on-replica-write`.




With `close-on-replica-write` enabled, any time a client attempts to write to a read-replica, the client connection to the cluster is closed. Your application logic should detect the disconnection, check the DNS table, and reconnect to the primary endpoint, which now would be endpoint B.




**When you might disable close-on-replica-write**

If disabling `close-on-replica-write` results in writes to the failing cluster, why disable `close-on-replica-write`?

As previously mentioned, with `close-on-replica-write` enabled, any time a client attempts to write to a read-replica the client connection to the cluster is closed. Establishing a new connection to the node takes time. Thus, disconnecting and reconnecting as a result of a write request to the replica also affects the latency of read requests that are served through the same connection. This effect remains in place until a new connection is established. If your application is especially read-heavy or very latency-sensitive, you might keep your clients connected to avoid degrading read performance.

### Redis OSS 2.8.22 (enhanced) added parameters

**Parameter group family:** redis2.8

For Redis OSS 2.8.22 there are no additional parameters supported.

> **Important**
>
> - Beginning with Redis OSS version 2.8.22, `repl-backlog-size` applies to the primary cluster as well as to replica clusters.
> - Beginning with Redis OSS version 2.8.22, the `repl-timeout` parameter is not supported. If it is changed, ElastiCache will overwrite with the default (60s), as we do with `appendonly`.

The following parameters are no longer supported.

- *appendonly*
- *appendfsync*
- *repl-timeout*

### Redis OSS 2.8.21 added parameters

**Parameter group family:** redis2.8

For Redis OSS 2.8.21, there are no additional parameters supported.

### Redis OSS 2.8.19 added parameters

**Parameter group family:** redis2.8

For Redis OSS 2.8.19 there are no additional parameters supported.

### Redis OSS 2.8.6 added parameters

**Parameter group family:** redis2.8

For Redis OSS 2.8.6 the following additional parameters are supported.

| Name | Details | Description |
| --- | --- | --- |
| `min-slaves-max-lag` | Default: 10<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Im ly | The number of seconds within which the primary node must receive a ping request from a read replica. If this amount of time passes and the primary does not receive a ping, then the replica is no longer considered available. If the number of available replicas drops below min-slaves-to-writ e, then the primary will stop accepting writes at that point.<br><br>If either this parameter or min-slaves-to-write is 0, then the primary node will always accept writes requests, even if no replic as are available. |
| `min-slaves-to-write` | Default: 0<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Im ly | The minimum number of read replicas which must be available in order for the primary node to accept writes from clients. If the number of available replicas falls below this number, then the primary node will no longer accept write requests.<br><br>If either this parameter or min-slaves-max-lag is 0, then the primary node will always accept writes requests, even if no replic as are available. |
| `notify-keyspace-events` | Default: (an empty string) | The types of keyspace events that Redis OSS can notify clients of. |

| Name | Details | Description |
| --- | --- | --- |
| | Type: string<br><br>Modifiable: Yes<br><br>Changes Take Effect: Im ly | Each event type is represented by a single letter:<br><br>• **K** — Keyspace events, published with a prefix of *__keyspace@<db>__*<br>• **E** — Key-event events, published with a prefix of *__keyevent@<db>__*<br>• **g** — Generic, non-specific commands such as *DEL*, *EXPIRE*, *RENAME*, etc.<br>• **$** — String commands<br>• **l** — List commands<br>• **s** — Set commands<br>• **h** — Hash commands<br>• **z** — Sorted set commands<br>• **x** — Expired events (events generated every time a key exp ires)<br>• **e** — Evicted events (events generated when a key is evicted for maxmemory)<br>• **A** — An alias for *g$lshzxe* |

| Name | Details | Description |
| --- | --- | --- |
| | | **Note:** Include at least **K** or **E** in the string so that Amazon ElastiCache can deliver notifications. Event-type characters (such as x for expired events) identify what happened, but do not trigger delivery on their own. If you omit **K** and **E**, Amazon ElastiCache accepts the configuration without error, but silently disables notifications.<br><br>You can have any combination of these event types. For example, *AKE* means that Redis OSS can publish notifications of all event types.<br><br>Do not use any characters other than those listed above; attempts to do so will result in error messages.<br><br>By default, this parameter is set to an empty string, meaning that keyspace event notification is disabled. |

| Name | Details | Description |
| --- | --- | --- |
| `repl-backlog-size` | Default: 1048576<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Im ly | The size, in bytes, of the primary node backlog buffer. The backlog is used for recording updates to data at the primary node. When a read replica connects to the primary, it attempts to perform a partial sync (`psync`), where it applies data from the backlog to catch up with the primary node. If the `psync` fails, then a full sync is required.<br><br>The minimum value for this parameter is 16384.<br><br>**Note**<br>Beginning with Redis OSS 2.8.22, this parameter applies to the primary cluster as well as the read replicas. |

| Name | Details | Description |
| --- | --- | --- |
| `repl-backlog-ttl` | Default: 3600<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Im ly | The number of seconds that the primary node will retain the backlog buffer. Starting from the time the last replica node disconnected, the data in the backlog will remain intact until `repl-backlog-ttl` expires. If the replica has not connected to the primary within this time, then the primary will release the backlog buffer. When the replica eventually reconnects, it will have to perform a full sync with the primary.<br><br>If this parameter is set to 0, then the backlog buffer will never be released. |
| `repl-timeout` | Default: 60<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Im ly | Represents the timeout period, in seconds, for:<br><br>• Bulk data transfer during synchronization, from the read replica's perspective<br>• Primary node timeout from the replica's perspective<br>• Replica timeout from the primary node's perspective |

**Redis OSS 2.6.13 parameters**

**Parameter group family:** redis2.6

Redis OSS 2.6.13 was the first version of Redis OSS supported by ElastiCache. The following table shows the Redis OSS 2.6.13 parameters that ElastiCache supports.

| Name | Details | Description |
| --- | --- | --- |
| `activerehashing` | Default: yes<br><br>Type: string (yes/no)<br><br>Modifiable: Yes<br><br>Changes take place:<br>At Creation | Determines whether to enable Redis' active rehashing feature. The main hash table is rehashed ten times per second; each rehash operation consumes 1 millisecond of CPU time.<br><br>This value is set when you create the parameter group. When assigning a new parameter group to a cluster, this value must be the same in both the old and new parameter groups. |
| `appendonly` | Default: no<br><br>Type: string<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Enables or disables Redis' append only file feature (AOF). AOF captures any Redis OSS commands that change data in the cache, and is used to recover from certain node failures.<br><br>The default value is *no*, meaning AOF is turned off. Set this parameter to *yes* to enable AOF.<br><br>For more information, see Mitigating Failures.<br><br>**Note**<br>For Multi-AZ replication groups, AOF is not allowed. |
| `appendfsync` | Default: everysec<br><br>Type: string<br><br>Modifiable: Yes | When `appendonly` is set to yes, controls how often the AOF output buffer is written to disk:<br><br>• |

| Name | Details | Description |
| --- | --- | --- |
| | Changes Take Effect: Immediately | *no* — the buffer is flushed to disk on an as-needed basis.<br>• *everysec* — the buffer is flushed once per second. This is the default.<br>• *always* — the buffer is flushed every time that data in the cluster is modified.<br>• Appendfsync is not supported for versions 2.8.22 and later. |
| `client-output-buffer-limit-normal-hard-limit` | Default: 0<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | If a client's output buffer reaches the specified number of bytes, the client will be disconnected. The default is zero (no hard limit). |
| `client-output-buffer-limit-normal-soft-limit` | Default: 0<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | If a client's output buffer reaches the specified number of bytes, the client will be disconnected, but only if this condition persists for `client-output-buffer-limit-normal-soft-seconds`. The default is zero (no soft limit). |
| `client-output-buffer-limit-normal-soft-seconds` | Default: 0<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | If a client's output buffer remains at `client-output-buffer-limit-normal-soft-limit` bytes for longer than this number of seconds, the client will be disconnected. The default is zero (no time limit). |

| Name | Details | Description |
| --- | --- | --- |
| client-output-buffer-limit-pubsub-hard-limit | Default: 33554432<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | For Redis OSS publish/subscribe clients: If a client's output buffer reaches the specified number of bytes, the client will be disconnected. |
| client-output-buffer-limit-pubsub-soft-limit | Default: 8388608<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | For Redis OSS publish/subscribe clients: If a client's output buffer reaches the specified number of bytes, the client will be disconnected, but only if this condition persists for client-output-buffer-limit-pubsub-soft-seconds . |
| client-output-buffer-limit-pubsub-soft-seconds | Default: 60<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | For Redis OSS publish/subscribe clients: If a client's output buffer remains at client-output-buffer-limit-pubsub-soft-limit bytes for longer than this number of seconds, the client will be disconnected. |
| client-output-buffer-limit-slave-hard-limit | Default: For values see Redis OSS node-type specific parameters<br>Type: integer<br>Modifiable: No | For Redis OSS read replicas: If a client's output buffer reaches the specified number of bytes, the client will be disconnected. |

| Name | Details | Description |
| --- | --- | --- |
| client-output-buffer-limit-slave-soft-limit | Default: For values see Redis OSS node-type specific parameters<br><br>Type: integer<br><br>Modifiable: No | For Redis OSS read replicas: If a client's output buffer reaches the specified number of bytes, the client will be disconnected, but only if this condition persists for client-output-buffer-limit-slave-soft-seconds . |
| client-output-buffer-limit-slave-soft-seconds | Default: 60<br><br>Type: integer<br><br>Modifiable: No | For Redis OSS read replicas: If a client's output buffer remains at client-output-buffer-limit-slave-soft-limit bytes for longer than this number of seconds, the client will be disconnected. |
| databases | Default: 16<br><br>Permitted values: 1 to 1200000<br><br>Type: integer<br><br>Modifiable: No<br><br>Changes take place: At Creation | The number of logical partitions the databases is split into. We recommend keeping this value low.<br><br>This value is set when you create the parameter group. When assigning a new parameter group to a cluster, this value must be the same in both the old and new parameter groups. |
| hash-max-ziplist-entries | Default: 512<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Determines the amount of memory used for hashes. Hashes with fewer than the specified number of entries are stored using a special encoding that saves space. |

| Name | Details | Description |
| --- | --- | --- |
| `hash-max-ziplist-value` | Default: 64<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | Determines the amount of memory used for hashes. Hashes with entries that are smaller than the specified number of bytes are stored using a special encoding that saves space. |
| `list-max-ziplist-entries` | Default: 512<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | Determines the amount of memory used for lists. Lists with fewer than the specified number of entries are stored using a special encoding that saves space. |
| `list-max-ziplist-value` | Default: 64<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | Determines the amount of memory used for lists. Lists with entries that are smaller than the specified number of bytes are stored using a special encoding that saves space. |
| `lua-time-limit` | Default: 5000<br>Type: integer<br>Modifiable: No | The maximum execution time for a Lua script, in milliseconds, before ElastiCache takes action to stop the script.<br>If `lua-time-limit` is exceeded, all Redis OSS commands will return an error of the form ____*-BUSY*. Since this state can cause interference with many essential Redis OSS operations, ElastiCache will first issue a *SCRIPT KILL* command. If this is unsuccessful, ElastiCache will forcibly restart Redis OSS. |

| Name | Details | Description |
| --- | --- | --- |
| `maxclients` This value applies to all instance types except those explicity specified | Default: 65000<br>Type: integer<br>Modifiable: No | The maximum number of clients that can be connected at one time. |
| | t4g.micro Default: 20000<br>Type: integer<br>Modifiable: No | |
| | t3.medium Default: 46000<br>Type: integer<br>Modifiable: No | |
| | t3.small Default: 46000<br>Type: integer<br>Modifiable: No | |
| | t3.micro Default: 20000<br>Type: integer<br>Modifiable: No | |
| `maxmemory-policy` | Default: volatile-lru<br>Type: string<br>Modifiable: Yes<br>Changes Take Effect: Immediately | The eviction policy for keys when maximum memory usage is reached.<br>Valid values are: `volatile-lru \| allkeys-lru \| volatile-random \| allkeys-random \| volatile-ttl \| noeviction`<br>For more information, see Using Valkey or Redis OSS as an LRU cache. |

| Name | Details | Description |
| --- | --- | --- |
| `maxmemory-samples` | Default: 3<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | For least-recently-used (LRU) and time-to-live (TTL) calculations, this parameter represents the sample size of keys to check. By default, Redis OSS chooses 3 keys and uses the one that was used least recently. |
| `reserved-memory` | Default: 0<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | The total memory, in bytes, reserved for non-data usage. By default, the Redis OSS node will grow until it consumes the node's `maxmemory` (see Redis OSS node-type specific parameters). If this occurs, then node performance will likely suffer due to excessive memory paging. By reserving memory you can set aside some of the available memory for non-Redis OSS purposes to help reduce the amount of paging.<br><br>This parameter is specific to ElastiCache, and is not part of the standard Redis OSS distribution.<br><br>For more information, see `reserved-memory-percent` and Managing reserved memory for Valkey and Redis OSS. |
| `set-max-intset-entries` | Default: 512<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Determines the amount of memory used for certain kinds of sets (strings that are integers in radix 10 in the range of 64 bit signed integers). Such sets with fewer than the specified number of entries are stored using a special encoding that saves space. |

| Name | Details | Description |
| --- | --- | --- |
| `slave-allow-chaining` | Default: no<br>Type: string<br>Modifiable: No | Determines whether a read replica in Redis OSS can have read replicas of its own. |
| `slowlog-log-slower-than` | Default: 10000<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | The maximum execution time, in microseconds, for commands to be logged by the Redis OSS Slow Log feature. |
| `slowlog-max-len` | Default: 128<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | The maximum length of the Redis OSS Slow Log. |
| `tcp-keepalive` | Default: 0<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: Immediately | If this is set to a nonzero value (N), node clients are polled every N seconds to ensure that they are still connected. With the default setting of 0, no such polling occurs.<br>**Important**<br>Some aspects of this parameter changed in Redis OSS version 3.2.4. See Parameters changed in Redis OSS 3.2.4 (enhanced). |

| Name | Details | Description |
| --- | --- | --- |
| `timeout` | Default: 0<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | The number of seconds a node waits before timing out. Values are:<br><br>• `0` – never disconnect an idle client.<br>• `1-19` – invalid values.<br>• `>=20` – the number of seconds a node waits before disconnecting an idle client. |
| `zset-max-ziplist-entries` | Default: 128<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Determines the amount of memory used for sorted sets. Sorted sets with fewer than the specified number of elements are stored using a special encoding that saves space. |
| `zset-max-ziplist-value` | Default: 64<br><br>Type: integer<br><br>Modifiable: Yes<br><br>Changes Take Effect: Immediately | Determines the amount of memory used for sorted sets. Sorted sets with entries that are smaller than the specified number of bytes are stored using a special encoding that saves space. |

**Note**

If you do not specify a parameter group for your Redis OSS 2.6.13 cluster, then a default parameter group (`default.redis2.6`) will be used. You cannot change the values of any parameters in the default parameter group; however, you can always create a custom parameter group and assign it to your cluster at any time.

### Valkey node-type specific parameters

The value of `maxmemory` is the maximum number of bytes available for data storage on the node. For more information, see Available memory.

> **Note**
>
> The `maxmemory` parameter cannot be modified.

| Node type | Maxmemory (bytes) |
|---|---|
| cache.t3.micro | 536870912 |
| cache.t3.small | 1471026299 |
| cache.t3.medium | 3317862236 |
| cache.t4g.micro | 536870912 |
| cache.t4g.small | 1471026299 |
| cache.t4g.medium | 3317862236 |
| cache.m4.large | 6892593152 |
| cache.m4.xlarge | 15328501760 |
| cache.m4.2xlarge | 31889126359 |
| cache.m4.4xlarge | 65257290629 |
| cache.m4.10xlarge | 166047614239 |
| cache.m5.large | 6854542746 |
| cache.m5.xlarge | 13891921715 |
| cache.m5.2xlarge | 27966669210 |
| cache.m5.4xlarge | 56116178125 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.m5.12xlarge | 168715971994 |
| cache.m5.24xlarge | 337500562842 |
| cache.m6g.large | 6854542746 |
| cache.m6g.xlarge | 13891921715 |
| cache.m6g.2xlarge | 27966669210 |
| cache.m6g.4xlarge | 56116178125 |
| cache.m6g.8xlarge | 111325552312 |
| cache.m6g.12xlarge | 168715971994 |
| cache.m6g.16xlarge | 225000375228 |
| cache.m7g.large | 6854542746 |
| cache.m7g.xlarge | 13891921715 |
| cache.m7g.2xlarge | 27966669210 |
| cache.m7g.4xlarge | 56116178125 |
| cache.m7g.8xlarge | 111325552312 |
| cache.m7g.12xlarge | 168715971994 |
| cache.m7g.16xlarge | 225000375228 |
| cache.m8g.large | 7294178820 |
| cache.m8g.xlarge | 15222495510 |
| cache.m8g.2xlarge | 31866224640 |
| cache.m8g.4xlarge | 65856864256 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.m8g.8xlarge | 133838143488 |
| cache.m8g.12xlarge | 201819422720 |
| cache.m8g.16xlarge | 269800701952 |
| cache.r4.large | 13201781556 |
| cache.r4.xlarge | 26898228839 |
| cache.r4.2xlarge | 54197537997 |
| cache.r4.4xlarge | 108858546586 |
| cache.r4.8xlarge | 218255432090 |
| cache.r4.16xlarge | 437021573120 |
| cache.r5.large | 14037181030 |
| cache.r5.xlarge | 28261849702 |
| cache.r5.2xlarge | 56711183565 |
| cache.r5.4xlarge | 113609865216 |
| cache.r5.12xlarge | 341206346547 |
| cache.r5.24xlarge | 682485973811 |
| cache.r6g.large | 14037181030 |
| cache.r6g.xlarge | 28261849702 |
| cache.r6g.2xlarge | 56711183565 |
| cache.r6g.4xlarge | 113609865216 |
| cache.r6g.8xlarge | 225000375228 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.r6g.12xlarge | 341206346547 |
| cache.r6g.16xlarge | 450000750456 |
| cache.r6gd.xlarge | 28261849702 |
| cache.r6gd.2xlarge | 56711183565 |
| cache.r6gd.4xlarge | 113609865216 |
| cache.r6gd.8xlarge | 225000375228 |
| cache.r6gd.12xlarge | 341206346547 |
| cache.r6gd.16xlarge | 450000750456 |
| cache.r7gd.xlarge | 31866224640 |
| cache.r7gd.2xlarge | 65856864256 |
| cache.r7gd.4xlarge | 133838143488 |
| cache.r7gd.8xlarge | 269800701952 |
| cache.r7gd.12xlarge | 405763260416 |
| cache.r7gd.16xlarge | 541725818880 |
| cache.r8gd.xlarge | 31866224640 |
| cache.r8gd.2xlarge | 65856864256 |
| cache.r8gd.4xlarge | 133838143488 |
| cache.r8gd.8xlarge | 269800701952 |
| cache.r8gd.12xlarge | 405763260416 |
| cache.r8gd.16xlarge | 541725818880 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.r7g.large | 14037181030 |
| cache.r7g.xlarge | 28261849702 |
| cache.r7g.2xlarge | 56711183565 |
| cache.r7g.4xlarge | 113609865216 |
| cache.r7g.8xlarge | 225000375228 |
| cache.r7g.12xlarge | 341206346547 |
| cache.r7g.16xlarge | 450000750456 |
| cache.r8g.large | 15222495510 |
| cache.r8g.xlarge | 31866224640 |
| cache.r8g.2xlarge | 65856864256 |
| cache.r8g.4xlarge | 133838143488 |
| cache.r8g.8xlarge | 269800701952 |
| cache.r8g.12xlarge | 405763260416 |
| cache.r8g.16xlarge | 541725818880 |
| cache.c7gn.large | 3317862236 |
| cache.c7gn.xlarge | 6854542746 |
| cache.c7gn.2xlarge | 13891921715 |
| cache.c7gn.4xlarge | 27966669210 |
| cache.c7gn.8xlarge | 56116178125 |
| cache.c7gn.12xlarge | 84357985997 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.c7gn.16xlarge | 113609865216 |
| cache.c8gn.large | 3330020474 |
| cache.c8gn.xlarge | 7294178820 |
| cache.c8gn.2xlarge | 15222495510 |
| cache.c8gn.4xlarge | 31866224640 |
| cache.c8gn.8xlarge | 65856864256 |
| cache.c8gn.12xlarge | 99847503872 |
| cache.c8gn.16xlarge | 133838143488 |

### Redis OSS node-type specific parameters

The value of `maxmemory` is the maximum number of bytes available for data storage on the node. For more information, see Available memory.

> **Note**
>
> The `maxmemory` parameter cannot be modified.

| Node type | Maxmemory (bytes) |
|---|---|
| cache.t3.micro | 536870912 |
| cache.t3.small | 1471026299 |
| cache.t3.medium | 3317862236 |
| cache.t4g.micro | 536870912 |
| cache.t4g.small | 1471026299 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.t4g.medium | 3317862236 |
| cache.m4.large | 6892593152 |
| cache.m4.xlarge | 15328501760 |
| cache.m4.2xlarge | 31889126359 |
| cache.m4.4xlarge | 65257290629 |
| cache.m4.10xlarge | 166047614239 |
| cache.m5.large | 6854542746 |
| cache.m5.xlarge | 13891921715 |
| cache.m5.2xlarge | 27966669210 |
| cache.m5.4xlarge | 56116178125 |
| cache.m5.12xlarge | 168715971994 |
| cache.m5.24xlarge | 337500562842 |
| cache.m6g.large | 6854542746 |
| cache.m6g.xlarge | 13891921715 |
| cache.m6g.2xlarge | 27966669210 |
| cache.m6g.4xlarge | 56116178125 |
| cache.m6g.8xlarge | 111325552312 |
| cache.m6g.12xlarge | 168715971994 |
| cache.m6g.16xlarge | 225000375228 |
| cache.m7g.large | 6854542746 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.m7g.xlarge | 13891921715 |
| cache.m7g.2xlarge | 27966669210 |
| cache.m7g.4xlarge | 56116178125 |
| cache.m7g.8xlarge | 111325552312 |
| cache.m7g.12xlarge | 168715971994 |
| cache.m7g.16xlarge | 225000375228 |
| cache.r4.large | 13201781556 |
| cache.r4.xlarge | 26898228839 |
| cache.r4.2xlarge | 54197537997 |
| cache.r4.4xlarge | 108858546586 |
| cache.r4.8xlarge | 218255432090 |
| cache.r4.16xlarge | 437021573120 |
| cache.r5.large | 14037181030 |
| cache.r5.xlarge | 28261849702 |
| cache.r5.2xlarge | 56711183565 |
| cache.r5.4xlarge | 113609865216 |
| cache.r5.12xlarge | 341206346547 |
| cache.r5.24xlarge | 682485973811 |
| cache.r6g.large | 14037181030 |
| cache.r6g.xlarge | 28261849702 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.r6g.2xlarge | 56711183565 |
| cache.r6g.4xlarge | 113609865216 |
| cache.r6g.8xlarge | 225000375228 |
| cache.r6g.12xlarge | 341206346547 |
| cache.r6g.16xlarge | 450000750456 |
| cache.r6gd.xlarge | 28261849702 |
| cache.r6gd.2xlarge | 56711183565 |
| cache.r6gd.4xlarge | 113609865216 |
| cache.r6gd.8xlarge | 225000375228 |
| cache.r6gd.12xlarge | 341206346547 |
| cache.r6gd.16xlarge | 450000750456 |
| cache.r7g.large | 14037181030 |
| cache.r7g.xlarge | 28261849702 |
| cache.r7g.2xlarge | 56711183565 |
| cache.r7g.4xlarge | 113609865216 |
| cache.r7g.8xlarge | 225000375228 |
| cache.r7g.12xlarge | 341206346547 |
| cache.r7g.16xlarge | 450000750456 |
| cache.c7gn.large | 3317862236 |
| cache.c7gn.xlarge | 6854542746 |

| Node type | Maxmemory (bytes) |
|---|---|
| cache.c7gn.2xlarge | 13891921715 |
| cache.c7gn.4xlarge | 27966669210 |
| cache.c7gn.8xlarge | 56116178125 |
| cache.c7gn.12xlarge | 84357985997 |
| cache.c7gn.16xlarge | 113609865216 |

**Note**

All current generation instance types are created in an Amazon Virtual Private Cloud VPC by default.

Redis OSS configuration variables `appendonly` and `appendfsync` are not supported.

### Memcached specific parameters

#### Memcached

If you do not specify a parameter group for your Memcached cluster, then a default parameter group appropriate to your engine version will be used. You can't change the values of any parameters in a default parameter group. However, you can create a custom parameter group and assign it to your cluster at any time. For more information, see Creating an ElastiCache parameter group.

#### Topics

- Memcached 1.6.17 changes
- Memcached 1.6.6 added parameters
- Memcached 1.5.10 parameter changes
- Memcached 1.4.34 added parameters
- Memcached 1.4.33 added parameters
- Memcached 1.4.24 added parameters
- Memcached 1.4.14 added parameters

- Memcached 1.4.5 supported parameters
- Memcached connection overhead
- Memcached node-type specific parameters

**Memcached 1.6.17 changes**

From Memcached 1.6.17, we no longer support these administrative commands: `lru_crawler`, `lru`, and `slabs`. With these changes, you will not be able to enable/disable `lru_crawler` at runtime via commands. Please enable/disable `lru_crawler` by modifying your custom parameter group.

**Memcached 1.6.6 added parameters**

For Memcached 1.6.6, no additional parameters are supported.

**Parameter group family:** memcached1.6

**Memcached 1.5.10 parameter changes**

For Memcached 1.5.10, the following additional parameters are supported.

**Parameter group family:** memcached1.5

| Name | Details | Description |
| --- | --- | --- |
| `no_modern` | Default: 1<br><br>Type: boolean<br><br>Modifiable: Yes<br><br>Allowed_Values: 0,1 | An alias for disabling `slab_reassign` , `lru_maintainer_thread` , `lru_segmented` , and `maxconns_fast` commands. |
| | Changes Take Effect: At launch | When using Memcached 1.5 and higher, `no_modern` also sets the hash_algorithm to `jenkins`.<br><br>In addition, when using Memcached 1.5.10, `inline_as` |

| Name | Details | Description |
| --- | --- | --- |
| | | cii_reponse is controlled by the parameter parallelly . This means that if no_modern is disabled then inline_as cii_reponse is disabled. From Memcached engine 1.5.16 onward the inline_as cii_response parameter no longer applies, so no_modern being abled or disabled has no effect on inline_as cii_reponse .<br><br>If no_modern is disabled, then slab_reassign , lru_maint ainer_thread , lru_segme nted , and maxconns_ fast WILL be enabled. Since slab_automove and hash_algorithm parameters are not SWITCH parameters, their setting is based on the configura tions in the parameter group.<br><br>If you want to disable no_modern and revert to modern, you must configure a custom parameter group to disable this parameter and then reboot for these changes to take effect.<br><br>**Note**<br>The default configuration value for this parameter |

| Name | Details | Description |
| --- | --- | --- |
| | | has been changed from 0 to 1 as of August 20, 2021. The updated default value will get automatically picked up by new ElastiCache users for each regions after August 20th, 2021. Existing ElastiCache users in the regions before August 20th, 2021 need to manually modify their custom parameter groups in order to pick up this new change. |
| `inline_ascii_resp` | Default: 0<br>Type: boolean<br>Modifiable: Yes<br>Allowed_Values: 0,1<br>Changes Take Effect: At launch | Stores numbers from `VALUE` response, inside an item, using up to 24 bytes. Small slowdown for ASCII `get`, `faster` sets. |

For Memcached 1.5.10, the following parameters are removed.

| Name | Details | Description |
| --- | --- | --- |
| `expirezero_does_not_evict` | Default: 0<br>Type: boolean<br>Modifiable: Yes | No longer supported in this version. |

| Name | Details | Description |
| --- | --- | --- |
| | Allowed_Values: 0,1<br><br>Changes Take Effect: At launch | |
| `modern` | Default: 1<br><br>Type: boolean<br><br>Modifiable: Yes (requires re-launch if set to `no_modern`)<br><br>Allowed_Values: 0,1<br><br>Changes Take Effect: At launch | No longer supported in this version. Starting with this version, `no-modern` is enabled by default with every launch or re-launch. |

**Memcached 1.4.34 added parameters**

For Memcached 1.4.34, no additional parameters are supported.

**Parameter group family:** memcached1.4

**Memcached 1.4.33 added parameters**

For Memcached 1.4.33, the following additional parameters are supported.

**Parameter group family:** memcached1.4

| Name | Details | Description |
| --- | --- | --- |
| `modern` | Default: enabled<br><br>Type: boolean<br><br>Modifiable: Yes<br><br>Changes Take Effect: At launch | An alias to multiple features. Enabling `modern` is equivalent to turning following commands on and using a `murmur3` hash algorithm: `slab_reassign`, `slab_automove`, |

| Name | Details | Description |
| --- | --- | --- |
| | | `lru_crawler` , `lru_maint ainer` , `maxconns_fast` , and `hash_algorithm=murmur3` . |
| `watch` | Default: enabled<br>Type: boolean<br>Modifiable: Yes<br>Changes Take Effect: Im ly<br>Logs can get dropped if user hits their `watcher_l ogbuf_size` and `worker_lo gbuf_size` limits. | Logs fetches, evictions or mutations. When, for example, user turns `watch` on, they can see logs when `get`, `set`, `delete`, or `update` occur. |
| `idle_timeout` | Default: 0 (disabled)<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: At | The minimum number of seconds a client will be allowed to idle before being asked to close. Range of values: 0 to 86400. |
| `track_sizes` | Default: disabled<br>Type: boolean<br>Modifiable: Yes<br>Changes Take Effect: At | Shows the sizes each slab group has consumed.<br>Enabling `track_sizes` lets you run `stats sizes` without the need to run `stats si zes_enable` . |

| Name | Details | Description |
| --- | --- | --- |
| watcher_logbuf_size | Default: 256 (KB)<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: At | The watch command turns on stream logging for Memcached. However watch can drop logs if the rate of evictions, mutations or fetches are high enough to cause the logging buffer to become full. In such situations, users can increase the buffer size to reduce the chance of log losse s. |
| worker_logbuf_size | Default: 64 (KB)<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: At | The watch command turns on stream logging for Memcached. However watch can drop logs if the rate of evictions, mutations or fetches are high enough to cause logging buffer get full. In such situations, users can increase the buffer size to reduce the chance of log losses. |
| slab_chunk_max | Default: 524288 (bytes)<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: At | Specifies the maximum size of a slab. Setting smaller slab size uses memory more efficiently. Items larger than slab_chun k_max are split over multiple slabs. |

| Name | Details | Description |
| --- | --- | --- |
| `lru_crawler metadump [all\|1\|2\|3]` | Default: disabled<br>Type: boolean<br>Modifiable: Yes<br>Changes Take Effect: Im ly | if lru_crawler is enabled this command dumps all keys.<br>`all\|1\|2\|3` - all slabs, or specify a particular slab number |

**Memcached 1.4.24 added parameters**

For Memcached 1.4.24, the following additional parameters are supported.

**Parameter group family:** memcached1.4

| Name | Details | Description |
| --- | --- | --- |
| `disable_flush_all` | Default: 0 (disabled)<br>Type: boolean<br>Modifiable: Yes<br>Changes Take Effect: At launch | Add parameter (`-F`) to disable flush_all. Useful if you never want to be able to run a full flush on production instances.<br>Values: 0, 1 (user can do a `flush_all` when the value is 0). |
| `hash_algorithm` | Default: jenkins<br>Type: string<br>Modifiable: Yes<br>Changes Take Effect: At launch | The hash algorithm to be used. Permitted values: murmur3 and jenkins. |
| `lru_crawler` | Default: 0 (disabled) | |

| Name | Details | Description |
| --- | --- | --- |
| | Type: boolean<br><br>Modifiable: Yes<br><br>Changes Take Effect: Af<br><br>**Note**<br>You can temporarily enable `lru_crawler` at runtime from the command line. For more information, see the Description column. | Cleans slab classes of items that have expired. This is a low impact process that runs in the background. Currently requires initiating a crawl using a manual command.<br><br>To temporarily enable, run `lru_crawler enable` at the command line.<br><br>`lru_crawler 1,3,5` crawls slab classes 1, 3, and 5 looking for expired items to add to the freelist.<br><br>Values: 0,1<br><br>**Note**<br>Enabling `lru_crawler` at the command line enables the crawler until either disabled at the command line or the next reboot. To enable permanently, you must modify the parameter value. For more information, see Modifying an ElastiCache parameter group. |

| Name | Details | Description |
| --- | --- | --- |
| `lru_maintainer` | Default: 0 (disabled)<br>Type: boolean<br>Modifiable: Yes<br>Changes Take Effect: At launch | A background thread that shuffles items between the LRUs as capacities are reached. Values: 0, 1. |
| `expirezero_does_not_evict` | Default: 0 (disabled)<br>Type: boolean<br>Modifiable: Yes<br>Changes Take Effect: At launch | When used with `lru_maintainer` , makes items with an expiration time of 0 unevictable.<br>**Warning**<br>This can crowd out memory available for other evictable items.<br>Can be set to disregard `lru_maintainer` . |

**Memcached 1.4.14 added parameters**

For Memcached 1.4.14, the following additional parameters are supported.

**Parameter group family:** memcached1.4

**Parameters added in Memcached 1.4.14**

| Name | Description |
| --- | --- |
| `config_max` | The maximum number of ElastiCache configuration entries. |

| Name | Description |
| --- | --- |
| | |
| config_size_max | The maximum size of the configuration entries, in bytes. |
| hashpower_init | The initial size of the ElastiCache hash table, expressed as a power of two. The default is 16 (2^16), or 65536 keys. |
| maxconns_fast | Changes the way in which new connections requests are handled when the maximum connection limit is reached. If this parameter is set to 0 (zero), new connections are added to the backlog queue and will wait until other connections are closed. If the parameter is set to 1, ElastiCache sends an error to the client and immediately closes the connection. |

| Name | Description |
| --- | --- |
| slab_automove | Adjusts the slab automove algorithm: If this parameter is set to 0 (zero), the automove algorithm is disabled. If it is set to 1, ElastiCache takes a slow, conservative approach to automatically moving slabs. If it is set to 2, ElastiCache aggressively moves slabs whenever there is an eviction. (This mode is not recommended except for testing purposes.) |
| slab_reassign | Enable or disable slab reassignment. If this parameter is set to 1, you can use the "slabs reassign" command to manually reassign memory. |

**Memcached 1.4.5 supported parameters**

**Parameter group family:** memcached1.4

For Memcached 1.4.5, the following parameters are supported.

**Parameters added in Memcached 1.4.5**

| Name | Details | Description |
| --- | --- | --- |
| backlog_queue_limit | Default: 1024<br><br>Type: integer | The backlog queue limit. |

| Name | Details | Description |
|---|---|---|
| | Modifiable: No | |
| binding_protocol | Default: auto<br>Type: string<br>Modifiable: Yes<br>Changes Take Effect: After resta | The binding protocol.<br>Permissible values are: ascii and auto.<br>For guidance on modifying the value of binding_protocol , see Modifying an ElastiCache parameter group. |
| cas_disabled | Default: 0 (false)<br>Type: Boolean<br>Modifiable: Yes<br>Changes Take Effect: After resta | If 1 (true), check and set (CAS) operations will be disabled, and items stored will consume 8 fewer bytes than with CAS enabled. |
| chunk_size | Default: 48<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: After resta | The minimum amount, in bytes, of space to allocate for the smallest item's key, value, and flags. |
| chunk_size_growth_factor | Default: 1.25<br>Type: float<br>Modifiable: Yes<br>Changes Take Effect: After resta | The growth factor that controls the size of each successive Memcached chunk; each chunk will be chunk_size_growth_factor times larger than the previous chunk. |
| error_on_memory_exhausted | Default: 0 (false)<br>Type: Boolean<br>Modifiable: Yes<br>Changes Take Effect: After resta | If 1 (true), when there is no more memory to store items, Memcached will return an error rather than evicting items. |

| Name | Details | Description |
| --- | --- | --- |
| large_memory_pages | Default: 0 (false)<br>Type: Boolean<br>Modifiable: No | If 1 (true), ElastiCache will try to use large memory pages. |
| lock_down_paged_memory | Default: 0 (false)<br>Type: Boolean<br>Modifiable: No | If 1 (true), ElastiCache will lock down all paged memory. |
| max_item_size | Default: 1048576<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: After resta | The size, in bytes, of the largest item that can be stored in the cluster. |
| max_simultaneous_connections | Default: 65000<br>Type: integer<br>Modifiable: No | The maximum number of simultaneous connections. |
| maximize_core_file_limit | Default: 0 (false)<br>Type: Boolean<br>Modifiable:<br>Changes Take Effect: After resta | If 1 (true), ElastiCache will maximize the core file limit. |
| memcached_connections_overhead | Default: 100<br>Type: integer<br>Modifiable: Yes<br>Changes Take Effect: After resta | The amount of memory to be reserved for Memcached connections and other miscellaneous overhead. For information about this parameter, see Memcached connection overhead. |

| Name | Details | Description |
| --- | --- | --- |
| requests_per_event | Default: 20<br>Type: integer<br>Modifiable: No | The maximum number of requests per event for a given connection. This limit is required to prevent resource starvation. |

**Memcached connection overhead**

On each node, the memory made available for storing items is the total available memory on that node (which is stored in the max_cache_memory parameter) minus the memory used for connections and other overhead (which is stored in the memcached_connections_overhead parameter). For example, a node of type cache.m5.large has a max_cache_memory of 6537MB. With the default memcached_connections_overhead value of 100MB, the Memcached process will have 6437MB available to store items.

The default values for the memcached_connections_overhead parameter satisfy most use cases; however, the required amount of allocation for connection overhead can vary depending on multiple factors, including request rate, payload size, and the number of connections.

You can change the value of the memcached_connections_overhead to better suit the needs of your application. For example, increasing the value of the memcached_connections_overhead parameter will reduce the amount of memory available for storing items and provide a larger buffer for connection overhead. Decreasing the value of the memcached_connections_overhead parameter will give you more memory to store items, but can increase your risk of swap usage and degraded performance. If you observe swap usage and degraded performance, try increasing the value of the memcached_connections_overhead parameter.

**Memcached node-type specific parameters**

Although most parameters have a single value, some parameters have different values depending on the node type used. The following table shows the default values for the max_cache_memory and num_threads parameters for each node type. The values on these parameters cannot be modified.

| Node type | max_cache_memory (in megabytes) | num_threads |
|---|---|---|
| cache.t3.micro | 512 | 2 |
| cache.t3.small | 1402 | 2 |
| cache.t3.medium | 3364 | 2 |
| cache.t4g.micro | 512 | 2 |
| cache.t4g.small | 1402 | 2 |
| cache.t4g.medium | 3164 | 2 |
| cache.m4.large | 6573 | 2 |
| cache.m4.xlarge | 11496 | 4 |
| cache.m4.2xlarge | 30412 | 8 |
| cache.m4.4xlarge | 62234 | 16 |
| cache.m4.10xlarge | 158355 | 40 |
| cache.m5.large | 6537 | 2 |
| cache.m5.xlarge | 13248 | 4 |
| cache.m5.2xlarge | 26671 | 8 |
| cache.m5.4xlarge | 53516 | 16 |
| cache.m5.12xlarge | 160900 | 48 |
| cache.m5.24xlarge | 321865 | 96 |
| cache.m6g.large | 6537 | 2 |
| cache.m6g.xlarge | 13248 | 4 |
| cache.m6g.2xlarge | 26671 | 8 |

| Node type | max_cache_memory (in megabytes) | num_threads |
|---|---|---|
| cache.m6g.4xlarge | 53516 | 16 |
| cache.m6g.8xlarge | 107000 | 32 |
| cache.m6g.12xlarge | 160900 | 48 |
| cache.m6g.16xlarge | 214577 | 64 |
| cache.m7g.large | 6537 | 2 |
| cache.m7g.xlarge | 13248 | 4 |
| cache.m7g.2xlarge | 26671 | 8 |
| cache.m7g.4xlarge | 53516 | 16 |
| cache.m7g.8xlarge | 107000 | 32 |
| cache.m7g.12xlarge | 160900 | 48 |
| cache.m7g.16xlarge | 214577 | 64 |
| cache.m8g.large | 6956 | 2 |
| cache.m8g.xlarge | 14517 | 4 |
| cache.m8g.2xlarge | 30390 | 8 |
| cache.m8g.4xlarge | 62806 | 16 |
| cache.m8g.8xlarge | 127638 | 32 |
| cache.m8g.12xlarge | 192470 | 48 |
| cache.m8g.16xlarge | 257302 | 64 |
| cache.r4.large | 12590 | 2 |
| cache.r4.xlarge | 25652 | 4 |

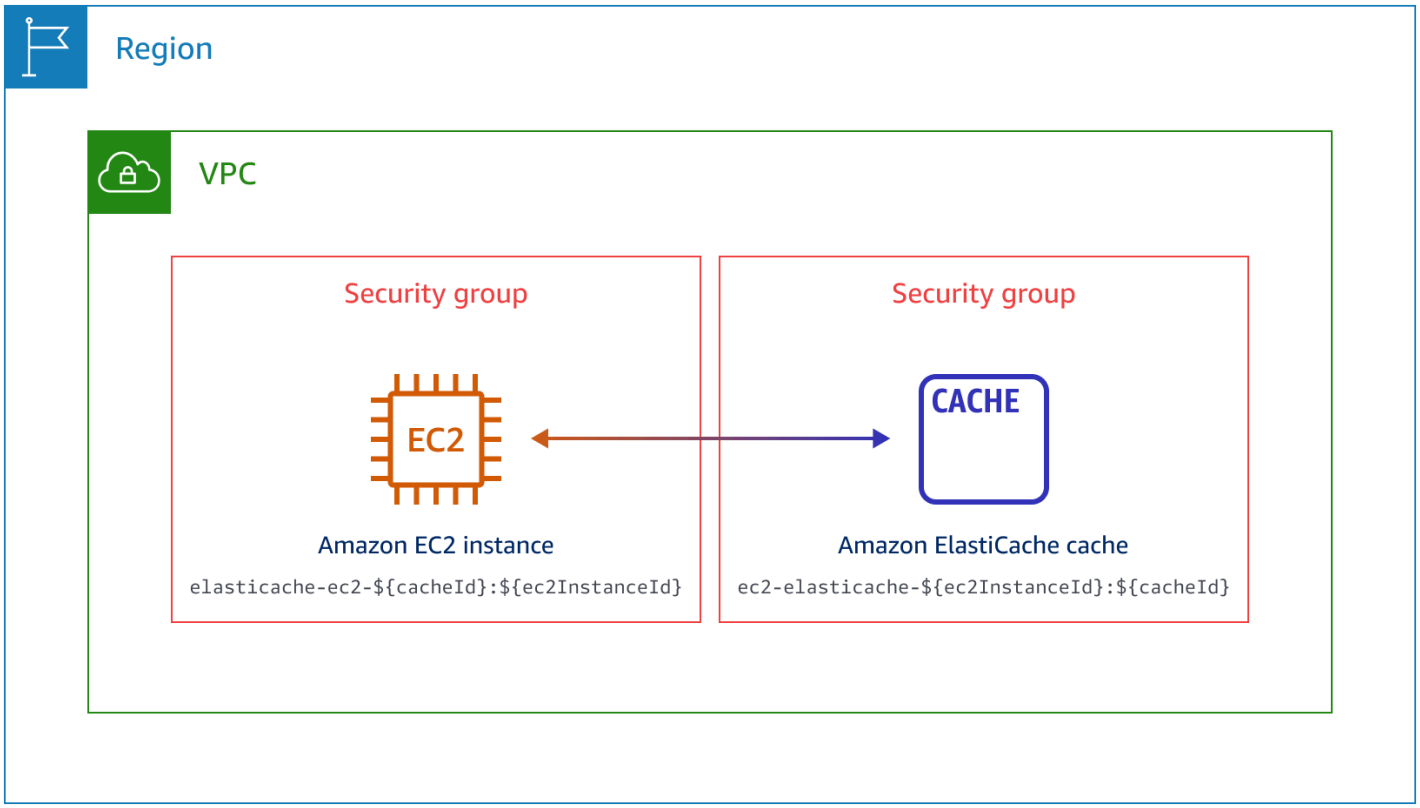
| Node type | max_cache_memory (in megabytes) | num_threads |
|---|---|---|
| cache.r4.2xlarge | 51686 | 8 |
| cache.r4.4xlarge | 103815 | 16 |
| cache.r4.8xlarge | 208144 | 32 |
| cache.r4.16xlarge | 416776 | 64 |
| cache.r5.large | 13387 | 2 |
| cache.r5.xlarge | 26953 | 4 |
| cache.r5.2xlarge | 54084 | 8 |
| cache.r5.4xlarge | 108347 | 16 |
| cache.r5.12xlarge | 325400 | 48 |
| cache.r5.24xlarge | 650869 | 96 |
| cache.r6g.large | 13387 | 2 |
| cache.r6g.xlarge | 26953 | 4 |
| cache.r6g.2xlarge | 54084 | 8 |
| cache.r6g.4xlarge | 108347 | 16 |
| cache.r6g.8xlarge | 214577 | 32 |
| cache.r6g.12xlarge | 325400 | 48 |
| cache.r6g.16xlarge | 429154 | 64 |
| cache.r7g.large | 13387 | 2 |
| cache.r7g.xlarge | 26953 | 4 |
| cache.r7g.2xlarge | 54084 | 8 |

| Node type | max_cache_memory (in megabytes) | num_threads |
|---|---|---|
| cache.r7g.4xlarge | 108347 | 16 |
| cache.r7g.8xlarge | 214577 | 32 |
| cache.r7g.12xlarge | 325400 | 48 |
| cache.r7g.16xlarge | 429154 | 64 |
| cache.r8g.large | 14517 | 2 |
| cache.r8g.xlarge | 30390 | 4 |
| cache.r8g.2xlarge | 62806 | 8 |
| cache.r8g.4xlarge | 127638 | 16 |
| cache.r8g.8xlarge | 257302 | 32 |
| cache.r8g.12xlarge | 386966 | 48 |
| cache.r8g.16xlarge | 516630 | 64 |
| cache.c7gn.large | 3164 | 2 |
| cache.c7gn.xlarge | 6537 | 4 |
| cache.c7gn.2xlarge | 13248 | 8 |
| cache.c7gn.4xlarge | 26671 | 16 |
| cache.c7gn.8xlarge | 53516 | 32 |
| cache.c7gn.12xlarge | 325400 | 48 |
| cache.c7gn.16xlarge | 108347 | 64 |
| cache.c8gn.large | 3176 | 2 |
| cache.c8gn.xlarge | 6956 | 4 |

| Node type | max_cache_memory (in megabytes) | num_threads |
|---|---|---|
| cache.c8gn.2xlarge | 14517 | 8 |
| cache.c8gn.4xlarge | 30390 | 16 |
| cache.c8gn.8xlarge | 62806 | 32 |
| cache.c8gn.12xlarge | 95222 | 48 |
| cache.c8gn.16xlarge | 127638 | 64 |

# Connecting an EC2 instance and an ElastiCache cache automatically

You can use the ElastiCache console to simplify setting up a connection between an Amazon Elastic Compute Cloud (Amazon EC2) instance and an ElastiCache cache. Often, your cache is in a private subnet and your EC2 instance is in a public subnet within a VPC. You can use a SQL client on your EC2 instance to connect to your ElastiCache cache. The EC2 instance can also run web servers or applications that access your private ElastiCache cache.




**Topics**


- Automatic connectivity with an EC2 instance
- Viewing connected compute resources


## Automatic connectivity with an EC2 instance

When you set up a connection between an EC2 instance and an ElastiCache cache, ElastiCache automatically configures the VPC security group for your EC2 instance and for your ElastiCache cache.

The following are requirements for connecting an EC2 instance with an ElastiCache cache:

- The EC2 instance must exist in the same VPC as the ElastiCache cache.

  If no EC2 instances exist in the same VPC, then the console provides a link to create one.
- The user who sets up connectivity must have permissions to perform the following Amazon EC2 operations. These permissiosn are generally added to EC2 accounts when they're created.

For more information on EC2 permissions, see Granting required permissions for Amazon EC2 resources.

- `ec2:AuthorizeSecurityGroupEgress`
- `ec2:AuthorizeSecurityGroupIngress`
- `ec2:CreateSecurityGroup`
- `ec2:DescribeInstances`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DescribeSecurityGroups`
- `ec2:ModifyNetworkInterfaceAttribute`
- `ec2:RevokeSecurityGroupEgress`

When you set up a connection to an EC2 instance, ElastiCache acts according to the current configuration of the security groups associated with the ElastiCache cache and EC2 instance, as described in the following table.

| Current ElastiCache security group configuration | Current EC2 security group configuration | ElastiCache action |
|---|---|---|
| There are one or more security groups associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source. | There are one or more security groups associated with the EC2 instance with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the ElastiCache cache as the source. | ElastiCache takes no action.<br><br>A connection was already configured automatically between the EC2 instance and the ElastiCache cache. Because a connection already exists between the EC2 instance and the ElastiCache cache, the security groups aren't modified. |
| Either of the following conditions apply: | Either of the following conditions apply: | ELC action: create new security groups |

| Current ElastiCache security group configuration | Current EC2 security group configuration | ElastiCache action |
| --- | --- | --- |
| • There is no security group associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` .<br>• There are one or more security groups associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . However, ElastiCache can't use any of these security groups for the connection with the EC2 instance. ElastiCache can't use a security group that doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. ElastiCache also can't use a security group that has been modified. Examples of modifications include adding a rule or changing the port of an existing rule. | • There is no security group associated with the EC2 instance with a name that matches the pattern `ec2-elasticache-${ec2InstanceId}:${cacheId}` .<br>• There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-elasticache-${ec2InstanceId}:${cacheId}` . However, ElastiCache can't use any of these security groups for the connection with the ElastiCache cache. ElastiCache can't use a security group that doesn't have one outbound rule with the VPC security group of the ElastiCache cache as the source. ElastiCache also can't use a security group that has been modified. | |

| Current ElastiCache security group configuration | Current EC2 security group configuration | ElastiCache action |
| --- | --- | --- |
| There are one or more security groups associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source. | There are one or more security groups associated with the EC2 instance with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . However, ElastiCache can't use any of these security groups for the connection with the ElastiCache cache. ElastiCache can't use a security group that doesn't have one outbound rule with the VPC security group of the ElastiCache cache as the source. ElastiCache also can't use a security group that has been modified. | ELC action: create new security groups |

| Current ElastiCache security group configuration | Current EC2 security group configuration | ElastiCache action |
| --- | --- | --- |
| There are one or more security groups associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source. | A valid EC2 security group for the connection exists, but it is not associated with the EC2 instance. This security group has a name that matches the pattern `ec2-elasticache-${ec2InstanceId}:${cacheId}` . It hasn't been modified. It has only one outbound rule with the VPC security group of theElastiCache cache as the source. | ELC action: associate EC2 security group |

| Current ElastiCache security group configuration | Current EC2 security group configuration | ElastiCache action |
| --- | --- | --- |
| Either of the following conditions apply:<br>• There is no security group associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` .<br>• There are one or more security groups associated with the ElastiCache cache with a name that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}` . However, ElastiCache can't use any of these security groups for the connection with the EC2 instance. ElastiCache can't use a security group that doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. ElastiCache also can't use security group that has been modified. | There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-elasticache-${ec2InstanceId}:${cacheId}` . A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the ElastiCache cache as the source. | ELC action: create new security groups |

**ElastiCache action: create new security groups**

ElastiCache takes the following actions:

- Creates a new security group that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}`. This security group has an inbound rule with the VPC security group of the EC2 instance as the source. This security group is associated with the ElastiCache cache and allows the EC2 instance to access it.
- Creates a new security group that matches the pattern `elasticache-ec2-${cacheId}:${ec2InstanceId}`. This security group has an outbound rule with the VPC security group of the ElastiCache cache as the target. This security group is associated with the EC2 instance and allows the EC2 instance to send traffic to the ElastiCache cache.

**ElastiCache action: associate EC2 security group**

ElastiCache associates the valid, existing EC2 security group with the EC2 instance. This security group allows the EC2 instance to send traffic to the ElastiCache cache.

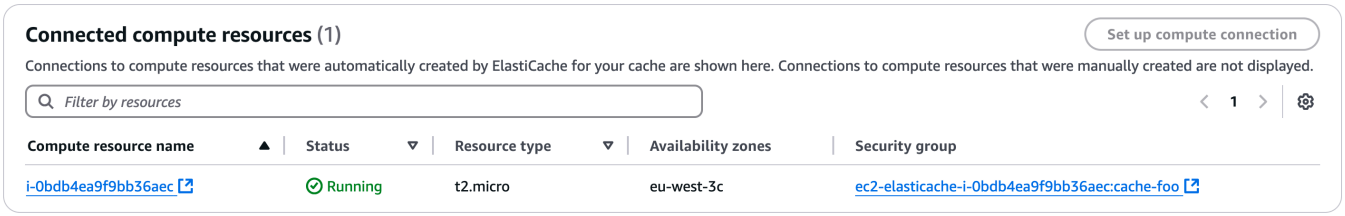
## Viewing connected compute resources

You can use the AWS Management Console to view the compute resources that are connected to an ElastiCache cache. The resources shown include compute resource connections that were set up automatically. For example, you can allow a compute resource to access a cache manually by adding a rule to the VPC security group associated with the cache. These resources will not appear in the connected compute resources list.

For a compute resource to be listed, the same conditions must apply as when automatically connecting an EC2 instance and an ElastiCache cache.

**To view compute resources connected to an ElastiCache cache**

1. Sign in to the AWS Management Console and open the ElastiCache console
2. In the navigation pane, choose **Caches**, and then choose a Valkey or Redis OSS cache.
3. On the **Connectivity & security** tab, view the compute resources in the **Set up compute connection**.



# Scaling ElastiCache

You can scale your ElastiCache cache to suit your needs. Serverless caches and node-based clusters offer several different scaling options.

- the section called “Considerations”
- the section called “Scaling ElastiCache Serverless clusters”
- the section called “Scaling node-based clusters”

## Considerations

### Potential impact on CPU utilization when scaling

When scaling up or down between node types, be aware of the potential impact on CPU utilization related to enhanced I/O features. For supported node types, ElastiCache offloads network I/O and TLS operations to dedicated threads by default, which utilize the extra CPU cores available on the node. The availability of these features depends on your engine version and node types:

- **Enhanced I/O (Redis 5.0.6+):** Network I/O is handled on dedicated threads, leveraging additional CPU cores on supported node types.
- **TLS offloading (Redis 6.2.5+):** TLS operations are offloaded to the I/O threads, further utilizing available CPU cores.
- **Enhanced I/O Multiplexing (Redis OSS 7.0.4+ or Valkey 7.2.6+):** Multiple client connections are multiplexed onto I/O threads, increasing throughput and optimizing CPU usage across available cores.

These features distribute processing across the extra CPU cores available on the node, which affects CPU metrics in the following ways:

#### Impact on CPUUtilization metric

`CPUUtilization` reflects the aggregate CPU usage across all cores on the node, including the dedicated I/O threads. Because enhanced I/O features consume CPU on these additional cores, `CPUUtilization` is not a reliable indicator of your engine's actual capacity and loads.

**Impact on EngineCPUUtilization metric**

`EngineCPUUtilization` measures only the main Redis or Valkey engine thread. When enhanced I/O features are active, operations such as network I/O and TLS processing are offloaded from the main thread to dedicated I/O threads. This means `EngineCPUUtilization` may decrease because the main thread is doing less work. `EngineCPUUtilization` accurately reflects your actual workload capacity and whether your instance is approaching its processing limits.

**Scaling scenarios**

- **Scaling from a non-supported to a supported node type:** When enhanced I/O features become active on the new node type, `CPUUtilization` may increase as dedicated I/O threads begin utilizing additional CPU cores. At the same time, `EngineCPUUtilization` may decrease as operations are offloaded from the main engine thread.
- **Scaling up within supported node types:** Additional CPU cores become available, which may reduce `CPUUtilization` as I/O operations are distributed across more resources.
- **Scaling down within supported node types:** Fewer CPU cores are available to handle I/O operations, which may increase `CPUUtilization` as network I/O, TLS processing, and connection handling compete for limited resources.

**Recommended monitoring approach**

We recommend using `EngineCPUUtilization` rather than `CPUUtilization` for monitoring. `EngineCPUUtilization` measures the main engine thread's performance and accurately reflects whether your instance is approaching its processing limits. `CPUUtilization` may vary across engine versions and node types due to changes in how enhanced I/O features utilize available cores, making it an unreliable metric for capacity planning.

## Scaling ElastiCache Serverless clusters

ElastiCache Serverless automatically accommodates your workload traffic as it ramps up or down. For each ElastiCache Serverless cache, ElastiCache continuously tracks the utilization of resources such as CPU, memory, and network. When any of these resources are constrained, ElastiCache Serverless scales out by adding a new shard and redistributing data to the new shard, without any downtime to your application. You can monitor the resources being consumed by your cache

in CloudWatch by monitoring the `BytesUsedForCache` metric for cache data storage and `ElastiCacheProcessingUnits` (ECPU) for compute usage.

## Setting scaling limits to manage costs

You can choose to configure a maximum usage on both cache data storage and ECPU/second for your cache to control cache costs. Doing so will ensure that your cache usage never exceeds the configured maximum.

If you set a scaling maximum then your application may experience decreased cache performance when the cache hits the maximum. When you set a cache data storage maximum and your cache data storage hits the maximum, ElastiCache will begin evicting data in your cache that has a Time-To-Live (TTL) set, using the LRU logic. If there is no data that can be evicted, then requests to write additional data will receive an Out Of Memory (OOM) error message. When you set an ECPU/second maximum and the compute utilization of your workload exceeds this value, ElastiCache will begin throttling requests.

If you setup a maximum limit on `BytesUsedForCache` or `ElastiCacheProcessingUnits`, we highly recommend setting up a CloudWatch alarm at a value lower than the maximum limit so that you are notified when your cache is operating close to these limits. We recommend setting an alarm at 75% of the maximum limit you set. See documentation about how to set up CloudWatch alarms.

## Pre-scaling with ElastiCache Serverless

### ElastiCache Serverless pre-scaling

With pre-scaling, also called pre-warming, you can set minimum supported limits for your ElastiCache cache. You can set these minimums for ElastiCache Processing Units (ECPUs) per second or data storage. This can be useful in preparation for anticipated scaling events. For example, if a gaming company expects a 5x increase in logins within the first minute that their new game launches, they can ready their cache for this significant spike in usage.

You can perform pre-scaling using the ElastiCache console, CLI, or API. ElastiCache Serverless updates the available ECPUs/second on the cache within 60 minutes, and sends an event notification when the minimum limit update is completed.

### How pre-scaling works

When the minimum limit for ECPUs/second or data storage is updated via the console, CLI, or API, that new limit is available within 1 hour. ElastiCache Serverless supports 30K ECPUs/second on

an empty cache, and up to 90K ECPUs/sec when using the Read from Replica feature. ElastiCache Serverless for Valkey 8.0 can double the supported requests per second (RPS) every 2-3 minutes, reaching 5M RPS per cache from zero in under 13 minutes, with consistent sub-millisecond p50 read latency. If you anticipate that an upcoming scaling event might exceed this rate, then we recommend setting the minimum ECPUs/second to the peak ECPUs/sec you expect at least 60 minutes before the peak event. Otherwise, the application may experience elevated latency and throttling of requests.

Once the minimum limit update is complete, ElastiCache Serverless will start metering you for the new minimum ECPUs per second or the new minimum storage. This occurs even if your application is not executing requests on the cache, or if your data storage usage is below the minimum. When you lower the minimum limit from its current setting, the update is immediate so ElastiCache Serverless will begin metering at the new minimum limit immediately.

**Note**

- When you set a minimum usage limit, you are charged for that limit even if your actual usage is lower than the minimum usage limit. ECPU or data storage usage that exceeds the minimum usage limit are charged the regular rate. For example, if you set a minimum usage limit of 100,000 ECPUs/second then you will be charged at least $1.224 per hour (using ECPU prices in us-east-1), even if your usage is lower than that set minimum.
- ElastiCache Serverless supports the requested minimum scale at an aggregate level on the cache. ElastiCache Serverless also supports a maximum of 30K ECPUs/second per slot (90K ECPUs/second when using Read from Replica using READONLY connections). As a best practice, your application should ensure that key distribution across Valkey or Redis OSS slots and traffic across keys is as uniform as possible.

## Setting scaling limits using the console and AWS CLI

*Setting scaling limits using the AWS Console*

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose the engine that runs on the cache that you want to modify.
3. A list of caches running the chosen engine appears.
4. Choose the cache to modify by choosing the radio button to the left of the cache's name.

5. Choose **Actions** and then choose **Modify**.
6. Under **Usage limits**, set appropriate **Memory** or **Compute** limits.
7. Click **Preview** changes and then **Save** changes.

#### Setting scaling limits using the AWS CLI

To change scaling limits using the CLI, use the modify-serverless-cache API.

**Linux:**

```
aws elasticache modify-serverless-cache --serverless-cache-name <cache name> \
--cache-usage-limits 'DataStorage={Minimum=10,Maximum=100,Unit=GB},
 ECPUPerSecond={Minimum=1000,Maximum=100000}'
```

**Windows:**

```
aws elasticache modify-serverless-cache --serverless-cache-name <cache name> ^
--cache-usage-limits 'DataStorage={Minimum=10,Maximum=100,Unit=GB},
 ECPUPerSecond={Minimum=1000,Maximum=100000}'
```

#### Removing scaling limits using the CLI

To remove scaling limits using the CLI, set the Minimum and Maximum limit parameters to 0.

**Linux:**

```
aws elasticache modify-serverless-cache --serverless-cache-name <cache name> \
--cache-usage-limits 'DataStorage={Minimum=0,Maximum=0,Unit=GB},
 ECPUPerSecond={Minimum=0,Maximum=0}'
```

**Windows:**

```
aws elasticache modify-serverless-cache --serverless-cache-name <cache name> ^
--cache-usage-limits 'DataStorage={Minimum=0,Maximum=0,Unit=GB},
 ECPUPerSecond={Minimum=0,Maximum=0}'
```

## Scaling node-based clusters

The amount of data your application needs to process is seldom static. It increases and decreases as your business grows or experiences normal fluctuations in demand. If you self-manage your

cache, you need to provision sufficient hardware for your demand peaks, which can be expensive. By using Amazon ElastiCache you can scale to meet current demand, paying only for what you use. ElastiCache enables you to scale your cache to match demand.

> **Note**
>
> If a Valkey or Redis OSS cluster is replicated across one or more Regions, then those Regions are scaled in order. When scaling up, secondary Regions are scaled first and then the primary Region. When scaling down, the primary Region is first and then any secondary Regions follow.
> When updating the engine version, the order is secondary Region and then primary Region.

**Topics**


- On-demand scaling for Memcached clusters
- Manual scaling for Memcached clusters
- Scaling for Valkey or Redis OSS (Cluster Mode Disabled) clusters
- Scaling replica nodes for Valkey or Redis OSS (Cluster Mode Disabled)
- Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters


## On-demand scaling for Memcached clusters

ElastiCache for Memcached offers a fully managed, in-memory caching service that deploys, operates, and vertically scales Memcached in the AWS cloud.

### On-demand vertical scaling

With vertical scaling, ElastiCache for Memcached provides a high-performance, distributed memory caching system widely used to speed up dynamic applications by alleviating database load. It stores data and objects in RAM, reducing the need to read from external data sources.

You can apply vertical scaling to existing node-based clusters as well as new ones. This can provide flexibility in resource allocation, allowing users to efficiently adapt to changing workloads without altering cluster architecture. This ability to scale improves performance by increasing cache capacity during high demand periods, and scaling down to optimize costs during low-demand periods. This simplifies operations, eliminates the need to create new clusters for shifting resource

requirements, and enables quick responses to traffic fluctuations. Overall, vertical scaling for Memcached node-based clusters can help enhance cost efficiency, improve resource utilization, and even let users change their Memcached instance type. All making it easier for users to align their caching infrastructure with actual application needs.

**Note**

- Node type modifications are only available for node-based Memcached clusters with engine versions 1.5 or later.
- Auto Discovery must be enabled in order to make use of vertical scaling.

### Setting up on-demand vertical scaling for node-based Memcached clusters

You can configure on-demand vertical scaling for Memcached with `scale-config`, which contains two parameters:

1. **ScaleIntervalMinutes:** Time (in minutes) between scaling batches during the Memcached upgrade process
2. **ScalePercentage:** Percentage of nodes to scale concurrently during the Memcached upgrade process

### Converting an existing Memcached node type to a cache that can vertically scale via the CLI

To convert an existing Memcached node-based cluster to a cache that can vertically scale, you can use `elasticache modify-cache-cluster` via the CLI.

```
aws elasticache modify-cache-cluster \
    --cache-cluster-id <your-cluster-id> \
    --cache-node-type <new-node-type> \
    --scale-config <scale-config> \
    --apply-immediately
```

### Setting up vertical scaling with the CLI

To set up vertical scaling for a node-based Memcached cluster via the CLI, use `elasticache modify-cache-cluster` with `scale-config` and its parameters `ScalePercentage` and `ScaleIntervalMinutes`.

- **scale-interval-minutes:**This defines the time (in minutes) between scaling batches. This setting can range from 2-30 minutes. If no value is specified, the default value of 5 minutes is applied.
- **scale-percentage:**This specifies the percentage of nodes to scale concurrently in each batch. This setting can range from 10-100. The setting is rounded up when dividing, so for example if the result would be 49.5 a setting of 50 is applied. If no value is specified, the default value of 20 is applied.

These configuration options will enable you to fine-tune the scaling process according to your specific needs, balancing between minimizing cluster disruption and optimizing scaling speed. The scale-config parameter will only be applicable for Memcached engine types and will be ignored for other cache engines, ensuring backward compatibility with existing API usage for other clusters.

**API call**

```
aws elasticache modify-cache-cluster \
    --cache-cluster-id <your-cluster-id> \
    --cache-node-type <new-node-type> \
    --scale-config '{
            "ScalePercentage": 30,
            "ScaleIntervalMinutes": 2
          }'
    --apply-immediately
```

**Result:**

Returns the cluster ID and the pending change.

```
{
    "CacheCluster": {
        "CacheNodeType": "old_insance_type",
         ...
         ...
         "PendingModifiedValues": {
            "CacheNodeType": "new_instance_type"
         },
    }
}
```

**List your Memcached cache vertical scaling setting**

You can retrieve scaling options for your Memcached caches, and see what their current options are for vertical scaling.

**API call**

```
aws elasticache list-allowed-node-type-modifications --cache-cluster-id <your-cluster-id>
```

**Result:**

```
{
  "ScaleUpModifications": [
      "cache.x.xxxx",
      "cache.x.xxxx"
      ],
   "ScaleDownModifications": [
      "cache.x.xxxx",
      "cache.x.xxxx",
      "cache.x.xxxx"
      ]
}
```

### Vertical scaling for Memcached with the AWS Management Console

Follow these steps to use the AWS Management Console to convert a node-based Memcached cluster to a vertically scalable cluster.

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. Select the Memcached cluster to convert.
3. Select the **Modify** tab.
4. Go to the **Cache settings** section, and select the desired **Node type**.
5. Select **Preview changes**, and review the changes.
6. Select **Modify**.

### Automated horizontal scaling for Memcached

ElastiCache now integrates with the AWS Application Auto Scaling (AAS) service to include automated horizontal scaling for Memcached clusters. You can define scaling policies through

the AWS Application Auto Scaling service, and automatically adjust the number of nodes in Memcached clusters as needed, based on predefined metrics or schedules.

> **Note**
>
> Automated horizontal scaling is not currently available in the Beijing and Ningxia Regions.

These are the available methods for automatically horizontally scaling your node-based clusters.

- **Scheduled Scaling:** Scaling based on a schedule allows you to set your own scaling schedule for predictable load changes. For example, every week the traffic to your web application starts to increase on Wednesday, remains high on Thursday, and starts to decrease on Friday. You can configure Auto Scaling to increase capacity on Wednesday and decrease capacity on Friday.
- **Target Tracking:** With target tracking scaling policies, you choose a scaling metric and set a target value. Application Auto Scaling creates and manages the CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and the target value. The scaling policy adds or removes capacity as required to keep the metric at, or close to, the specified target value.

#### How to set up horizontal scaling for a node-based Memcached cluster via the CLI

When horizontal scaling a node-based Memcached cluster, you can use a target tracking policy, a scheduled policy, or both.

1. **Register a resource as scalable target**

   Call the `RegisterScalableTarget` API in AWS Application Auto Scaling to register the target for the scalable dimension `elasticache:cache-cluster:Nodes`.

   **API: ApplicationAutoScaling.RegisterScalableTarget**

   Input:

```
{
 "ScalableDimension": "elasticache:cache-cluster:Nodes",
 "ResourceId": "cache-cluster/test-cluster-1",
 "ServiceNamespace": "elasticache",
 "MinCapacity": 20,
 "MaxCapacity": 50
```

```
}
```

2. **Create a Target tracking scaling policy**

   Next, you can create a target tracking scaling policy for the resource by calling put scaling policy API.

3. **Predefined Metric**

   Following is a policy that scales along the dimension of Cache Node, using the predefined metric `ElastiCacheCPUUtilization`, set at 50 for cluster test-cluster-1. When deleting nodes for scale-in, the last n nodes will be removed.

   API: ApplicationAutoScaling.PutScalingPolicy

   Input:

```
{
 "PolicyName": "cpu50-target-tracking-scaling-policy",
 "PolicyType": "TargetTrackingScaling",
 "TargetTrackingScalingPolicyConfiguration": {
  "TargetValue": 50,
  "PredefinedMetricSpecification": {
   "PredefinedMetricType": "ElastiCacheCPUUtilization"
   },
  "ScaleOutCooldown": 600,
  "ScaleInCooldown": 600
   },
 "ServiceNamespace": "elasticache",
 "ScalableDimension": "elasticache:cache-cluster:Nodes",
 "ResourceId": "cache-cluster/test-cluster-1"
}
```

   Output:

```
{
 "PolicyARN": "arn:aws:autoscaling:us-west-2:012345678910:scalingPolicy:6d8972f3-
efc8-437c-92d1-6270f29a66e7:resource/elasticache/cache-cluster/test-
cluster-1:policyName/cpu50-target-tracking-scaling-policy",
 "Alarms": [
  {
```

```
  "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-
elasticache/cache-cluster/test-cluster-1-AlarmHigh-d4f0770c-b46e-434a-
a60f-3b36d653feca",
  "AlarmName": "TargetTracking-elasticache/cache-cluster/test-cluster-1-AlarmHigh-
d4f0770c-b46e-434a-a60f-3b36d653feca"
  },
  {
  "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-
elasticache/cache-cluster/test-cluster-1-AlarmLow-1b437334-d19b-4a63-
a812-6c67aaf2910d",
  "AlarmName": "TargetTracking-elasticache/cache-cluster/test-cluster-1-
AlarmLow-1b437334-d19b-4a63-a812-6c67aaf2910d"
  }
 ]
}
```

4. **Custom Metric**

   You can also set scaling policy on dimension by using a custom percentage that's based on the Cloudwatch metric.

   Input:

```
{
 "PolicyName": "cpu50-target-tracking-scaling-policy",
 "PolicyType": "TargetTrackingScaling",
 "TargetTrackingScalingPolicyConfiguration": {
  "CustomizedMetricSpecification": {
   "Dimensions": [
    {
    "Name": "MyMetricDimension",
    "Value": "DimensionValue"
    }
    ],
   "MetricName": "MyCustomMetric",
   "Namespace": "MyNamespace",
   "Statistic": "Average",
   "Unit": "Percent"
   },
  "TargetValue": 40,
  "ScaleOutCooldown": 600,
  "ScaleInCooldown": 600
  },
```

```
 "ServiceNamespace": "elasticache",
 "ScalableDimension": "elasticache:cache-cluster:Nodes",
 "ResourceId": "cache-cluster/test-cluster-1"
}
```

5. **Scheduled Actions**

   When you need to scale out for a particular event and then scale in after the event, you can create two scheduled actions by calling the `PutScheduledAction` API.

   **Policy 1: Scaling out**

   The `at` command in `--schedule` schedules the action to be run once at a specified date and time in the future. The schedule field also supports rate (minute, hour, day etc) and cron (for cron expression).

   At the date and time specified, Application Auto Scaling updates the `MinCapacity` and `MaxCapacity` values. Application Auto Scaling scales out to MinCapacity to put the cache nodes to 70.

   **API: ApplicationAutoScaling.PutScheduledAction**

   Input:

```
{
 "ResourceId": "elasticache:ache-cluster:test-cluster-1",
 "ScalableDimension": "elasticache:cache-cluster:Nodes",
  "ScalableTargetAction": {
   "MaxCapacity": 100,
   "MinCapacity": 70
   },
 "Schedule": "at(2020-05-20T17:05:00)",
 "ScheduledActionName": "ScalingOutScheduledAction",
 "ServiceNamespace": "elasticache",
}
```

   **Policy 2: Scaling in**

   At the date and time specified, Application Auto Scaling updates the table's `MinCapacity` and `MaxCapacity`, and scales in to `MaxCapacity` to return the cache nodes to 60.

   **API: ApplicationAutoScaling.PutScheduledAction**

Input:

```
{
 "ResourceId": "elasticache:cache-cluster:test-cluster-1",
 "ScalableDimension": "elasticache:cache-cluster:Nodes",
 "ScalableTargetAction": {
  "MaxCapacity": 60,
  "MinCapacity": 40
  },
 "Schedule": "at(2020-05-21T17:05:00)",
 "ScheduledActionName": "ScalingInScheduledAction",
 "ServiceNamespace": "elasticache",
}
```

6. **View the Scaling Activities**

   You can view the scaling activities using the `DescribeScalingActivities` API.

   **API: ApplicationAutoScaling.DescribeScalingActivities**

   Output:

```
{
 "ScalingActivities": [
  {
  "ScalableDimension": "elasticache:elasticache:DesiredCount",
  "Description": "Setting desired count to 30.",
  "ResourceId": "elasticache/cache-cluster/test-cluster-1",
  "ActivityId": "4d759079-a31f-4d0c-8468-504c56e2eecf",
  "StartTime": 1462574194.658,
  "elasticacheNamespace": "elasticache",
  "EndTime": 1462574276.686,
  "Cause": "monitor alarm TargetTracking-elasticache/cache-cluster/test-cluster-1-
AlarmHigh-d4f0770c-b46e-434a-a60f-3b36d653feca in state ALARM triggered policy
 cpu50-target-tracking-scaling-policy",
  "StatusMessage": "Failed to set desired count to 30",
  "StatusCode": "Failed"
  },
  {
  "ScalableDimension": "elasticache:elasticache:DesiredCount",
  "Description": "Setting desired count to 25.",
  "ResourceId": "elasticache/cache-cluster/test-cluster-1",
```

```
  "ActivityId": "90aff0eb-dd6a-443c-889b-b809e78061c1",
  "StartTime": 1462574254.223,
  "elasticacheNamespace": "elasticache",
  "EndTime": 1462574333.492,
  "Cause": "monitor alarm TargetTracking-elasticache/cache-cluster/test-cluster-1-
AlarmHigh-d4f0770c-b46e-434a-a60f-3b36d653feca in state ALARM triggered policy
 cpu50-target-tracking-scaling-policy",
  "StatusMessage": "Successfully set desired count to 25. Change successfully
 fulfilled by elasticache.",
  "StatusCode": "Successful"
  }
 ]
}
```

7. **Edit/Delete Scaling Policy**

   You can edit or delete policies by calling `PutScalingPolicy` API again, or by calling `DeleteScalingPolicy` or `DeleteScheduled` Action.

8. **De-register scalable targets**

   You can de-register the scalable target through the `DeregisterScalableTarget` API. Deregistering a scalable target deletes the scaling policies and the scheduled actions that are associated with it.

   **API: ApplicationAutoScaling.DeregisterScalableTarget**

   Input:

```
{
 "ResourceId": "elasticache/cache-cluster/test-cluster-1",
 "ServiceNamespace": "elasticache",
 "ScalableDimension": "elasticache:cache-cluster:Nodes"
}
```

9. **Scaling Policy Cleanup**

10. **Multiple Scaling Policies**

    You can create multiple scaling policies. Following are key callouts on behavior from Auto scaling target tracking.

    - You can have multiple target tracking scaling policies for a scalable target, provided that each of them uses a different metric.

- The intention of Application Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the scalable target if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion enabled) are ready to scale in.
- If multiple policies instruct the scalable target to scale out or in at the same time, Application Auto Scaling scales based on the policy that provides the largest capacity for both scale in and scale out. This provides greater flexibility to cover multiple scenarios and ensures that there is always enough capacity to process your application workloads.

> **Note**
>
> AWS Application Auto Scaling does not queue scaling policies. Application Auto Scaling will wait for the first scaling to complete, then cooldown, and then repeat the above algorithm.

**Automatically horizontally scale a node-based Memcached cluster via the AWS Management Console**

Follow these steps to use the AWS Management Console to convert an existing node-based Memcached cluster to a horizontally scalable cluster.

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. Select the Memcached cache to convert.
3. Go to the **Autoscaling** tab.
4. Select the scaling policy to apply, by selecting either **Add dynamic scaling** or **Add scheduled scaling**.
5. Fill in the details for the selected policy as needed.
6. Click **Create**.

## Manual scaling for Memcached clusters

Manually horizontally scaling a Memcached cluster in or out is as easy as adding or removing nodes from the cluster. Memcached clusters are composed of 1 to 60 nodes.

Because you can partition your data across all the nodes in a Memcached cluster, scaling up to a node type with greater memory is seldom required. However, because the Memcached engine does not persist data, if you do scale to a different node type then your new cluster starts out empty unless your application populates it.

To manually vertically scale your Memcached cluster, you must create a new cluster. Memcached clusters always start out empty unless your application populates it.

**Manually scaling Memcached clusters**

| Action | Topic |
| --- | --- |
| Scaling out | Adding nodes to a cluster |
| Scaling in | Deleting nodes from a cluster |
| Changing node types | Manually scaling node-based Memcached clusters vertically |

**Topics**

- Manually scaling a node-based Memcached cluster horizontally
- Manually scaling node-based Memcached clusters vertically

**Manually scaling a node-based Memcached cluster horizontally**

The Memcached engine supports partitioning your data across multiple nodes. Because of this, Memcached clusters scale horizontally easily. To horizontally scale your Memcached cluster, merely add or remove nodes.

The following topics detail how to scale your Memcached cluster out or in by adding or removing nodes.

- Adding nodes to a cluster
- Deleting nodes from your cluster

Each time you change the number of nodes in your Memcached cluster, you must re-map at least some of your keyspace so it maps to the correct node. For more detailed information on load balancing your Memcached cluster, see Configuring your ElastiCache client for efficient load balancing (Memcached).

If you use auto discovery on your Memcached cluster, you do not need to change the endpoints in your application as you add or remove nodes. For more information on auto discovery, see Automatically identify nodes in your cluster (Memcached) If you do not use auto discovery, each time you change the number of nodes in your Memcached cluster you must update the endpoints in your application.

### Manually scaling node-based Memcached clusters vertically

When you manually scale your Memcached cluster up or down, you must create a new cluster. Memcached clusters always start out empty unless your application populates it.

> **Important**
>
> If you are scaling down to a smaller node type, be sure that the smaller node type is adequate for your data and overhead. For more information, see Choosing your node size.

#### Topics

- Scaling a node-based Memcached cluster vertically (Console)
- Scaling a node-based Memcached cluster vertically (AWS CLI)
- Scaling a node-based Memcached cluster vertically (ElastiCache API)

### Scaling a node-based Memcached cluster vertically (Console)

The following procedure walks you through scaling a node-based Memcached cluster vertically using the AWS Management Console.

1. Create a new cluster with the new node type. For more information, see Creating a Memcached cluster (console).
2. In your application, update the endpoints to the new cluster's endpoints. For more information, see Finding a Cluster's Endpoints (Console) (Memcached).
3. Delete the old cluster. For more information, see Deleting a new node in Memcached.

### Scaling a node-based Memcached cluster vertically (AWS CLI)

The following procedure walks you through scaling a node-based Memcached cluster vertically using the AWS CLI.

1. Create a new cluster with the new node type. For more information, see Creating a cluster (AWS CLI).
2. In your application, update the endpoints to the new cluster's endpoints. For more information, see Finding Endpoints (AWS CLI).
3. Delete the old cluster. For more information, see Using the AWS CLI to delete an ElastiCache cluster.

### Scaling a node-based Memcached cluster vertically (ElastiCache API)

The following procedure walks you through scaling a node-based Memcached cluster vertically using the ElastiCache API.

1. Create a new cluster with the new node type. For more information, see Creating a cluster for Memcached (ElastiCache API)
2. In your application, update the endpoints to the new cluster's endpoints. For more information, see Finding Endpoints (ElastiCache API).
3. Delete the old cluster. For more information, see Using the ElastiCache API.

# Scaling for Valkey or Redis OSS (Cluster Mode Disabled) clusters

Valkey or Redis OSS (cluster mode disabled) clusters can be a single-node cluster with 0 shards or multi-node clusters with 1 shard. Single-node clusters use the one node for both reads and writes. Multi-node clusters always have 1 node as the read/write primary node with 0 to 5 read-only replica nodes.

## Topics


- Scaling for Valkey or Redis OSS (Cluster Mode Disabled) clusters


### Scaling Valkey or Redis OSS clusters

| Action | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|
| Scaling in | Removing nodes from an ElastiCache cluster | Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters |
| Scaling out | Adding nodes to a cluster | Online resharding for Valkey or Redis OSS (cluster mode enabled) |
| Changing node types | To a larger node type:<br>• Scaling up single-node Valkey or Redis OSS clusters<br>• Scaling up Valkey or Redis OSS clusters with replicas<br><br>To a smaller node type:<br>• Scaling down single-node Valkey or Redis OSS clusters | Online vertical scaling by modifying node type |

| Action | Valkey or Redis OSS (cluster mode disabled) | Valkey or Redis OSS (cluster mode enabled) |
|---|---|---|
| | • Scaling down Valkey or Redis OSS clusters with replicas | |
| Changing the number of node groups | Not supported for Valkey or Redis OSS (cluster mode disabled) clusters | Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters |

**Contents**


- Scaling for Valkey or Redis OSS (Cluster Mode Disabled) clusters
  - Scaling up single-node Valkey or Redis OSS clusters
    - Scaling up single-node Valkey or Redis OSS (Cluster Mode Disabled) (Console) clusters
    - Scaling up single-node Valkey or Redis OSS clusters (AWS CLI)
    - Scaling up single-node Valkey or Redis OSS clusters (ElastiCache API)
  - Scaling down single-node Valkey or Redis OSS clusters
    - Scaling down a single-node Valkey or Redis OSS cluster (Console)
    - Scaling down single-node Valkey or Redis OSS clusters (AWS CLI)
    - Scaling down single-node Valkey or Redis OSS clusters (ElastiCache API)


## Scaling for Valkey or Redis OSS (Cluster Mode Disabled) clusters

Valkey or Redis OSS (cluster mode disabled) nodes must be large enough to contain all the cache's data plus the Valkey or Redis OSS overhead. To change the data capacity of your Valkey or Redis OSS (cluster mode disabled) cluster, you must scale vertically; scaling up to a larger node type to increase data capacity, or scaling down to a smaller node type to reduce data capacity.

The ElastiCache scaling up process is designed to make a best effort to retain your existing data and requires successful Valkey or Redis OSS replication. For Valkey or Redis OSS (cluster mode disabled) clusters, we recommend that sufficient memory be made available to Valkey or Redis OSS.

You cannot partition your data across multiple Valkey or Redis OSS (cluster mode disabled) clusters. However, if you only need to increase or decrease your cluster's read capacity, you can

create a Valkey or Redis OSS (cluster mode disabled) cluster with replica nodes and add or remove read replicas. To create a Valkey or Redis OSS (cluster mode disabled) cluster with replica nodes using your single-node Valkey or Redis OSS cluster as the primary cluster, see Creating a Valkey (cluster mode disabled) cluster (Console).

After you create the cluster with replicas, you can increase read capacity by adding read replicas. Later, if you need to, you can reduce read capacity by removing read replicas. For more information, see Increasing read capacity or Decreasing read capacity.

In addition to being able to scale read capacity, Valkey or Redis OSS (cluster mode disabled) clusters with replicas provide other business advantages. For more information, see High availability using replication groups.

**Important**

If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, before you begin scaling be sure that you have a custom parameter group that reserves the correct amount of memory for your new node type. Alternatively, you can modify a custom parameter group so that it uses `reserved-memory-percent` and use that parameter group for your new cluster.
If you're using `reserved-memory-percent`, doing this is not necessary.
For more information, see Managing reserved memory for Valkey and Redis OSS.

**Topics**

- Scaling up single-node Valkey or Redis OSS clusters
- Scaling down single-node Valkey or Redis OSS clusters

### Scaling up single-node Valkey or Redis OSS clusters

When you scale up a single-node Valkey or Redis OSS cluster, ElastiCache performs the following process, whether you use the ElastiCache console, the AWS CLI, or the ElastiCache API.

1. A new cluster with the new node type is spun up in the same Availability Zone as the existing cluster.
2. The cache data in the existing cluster is copied to the new cluster. How long this process takes depends upon your node type and how much data is in the cluster.
3. Reads and writes are now served using the new cluster. Because the new cluster's endpoints are the same as they were for the old cluster, you do not need to update the endpoints in your application. You will notice a brief interruption (a few seconds) of reads and writes from the primary node while the DNS entry is updated.
4. ElastiCache deletes the old cluster. You will notice a brief interruption (a few seconds) of reads and writes from the old node because the connections to the old node will be disconnected.

> **Note**
>
> For clusters running a data tiering node type, you can only scale to node sizes within the same data tiering node family.

As shown in the following table, your Valkey or Redis OSS scale-up operation is blocked if you have an engine upgrade scheduled for the next maintenance window. For more information on Maintenance Windows, see Managing ElastiCache cluster maintenance.

#### Blocked Valkey or Redis OSS operations

| Pending Operations | Blocked Operations |
|---|---|
| Scale up | Immediate engine upgrade |
| Engine upgrade | Immediate scale up |
| Scale up and engine upgrade | Immediate scale up |
| | Immediate engine upgrade |

If you have a pending operation that is blocking you, you can do one of the following.

- Schedule your Valkey or Redis OSS scale-up operation for the next maintenance window by clearing the **Apply immediately** check box (CLI use: `--no-apply-immediately`, API use: `ApplyImmediately=false`).
- Wait until your next maintenance window (or after) to perform your Valkey or Redis OSS scale up operation.
- Add the Valkey or Redis OSS engine upgrade to this cluster modification with the **Apply Immediately** check box chosen (CLI use: `--apply-immediately`, API use: `ApplyImmediately=true`). This unblocks your scale up operation by causing the engine upgrade to be performed immediately.

You can scale up a single-node Valkey or Redis OSS (cluster mode disabled) cluster using the ElastiCache console, the AWS CLI, or ElastiCache API.

> **Important**
>
> If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, before you begin scaling be sure that you have a custom parameter group that reserves the correct amount of memory for your new node type. Alternatively, you can modify a custom parameter group so that it uses `reserved-memory-percent` and use that parameter group for your new cluster.
> If you're using `reserved-memory-percent`, doing this is not necessary.
> For more information, see Managing reserved memory for Valkey and Redis OSS.

### Scaling up single-node Valkey or Redis OSS (Cluster Mode Disabled) (Console) clusters

The following procedure describes how to scale up a single-node Valkey or Redis OSS cluster using the ElastiCache Management Console. During this process, your Valkey or Redis OSS cluster will continue to serve requests with minimal downtime.

**To scale up a single-node Valkey or Redis OSS cluster (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey or Redis OSS clusters**.

3. From the list of clusters, choose the cluster you want to scale up (it must be running the Valkey or Redis OSS engine, not the clustered Valkey or Redis OSS engine).
4. Choose **Modify**.
5. In the **Modify Cluster** wizard:

   a. Choose the node type you want to scale to from the **Node type** list.

   b. If you're using `reserved-memory` to manage your memory, from the **Parameter Group** list, choose the custom parameter group that reserves the correct amount of memory for your new node type.
6. If you want to perform the scale up process right away, choose the **Apply immediately** box. If the **Apply immediately** box is not chosen, the scale-up process is performed during this cluster's next maintenance window.
7. Choose **Modify**.

   If you chose **Apply immediately** in the previous step, the cluster's status changes to *modifying*. When the status changes to *available*, the modification is complete and you can begin using the new cluster.

### Scaling up single-node Valkey or Redis OSS clusters (AWS CLI)

The following procedure describes how to scale up a single-node Valkey or Redis OSS cluster using the AWS CLI. During this process, your Valkey or Redis OSS cluster will continue to serve requests with minimal downtime.

### To scale up a single-node Valkey or Redis OSS cluster (AWS CLI)

1. Determine the node types you can scale up to by running the AWS CLI `list-allowed-node-type-modifications` command with the following parameter.

   - `--cache-cluster-id`

   For Linux, macOS, or Unix:

   ```
   aws elasticache list-allowed-node-type-modifications \
        --cache-cluster-id my-cache-cluster-id
   ```

   For Windows:

```
aws elasticache list-allowed-node-type-modifications ^
     --cache-cluster-id my-cache-cluster-id
```

Output from the above command looks something like this (JSON format).

```
{
     "ScaleUpModifications": [
         "cache.m5.2xlarge",
         "cache.m5.large",
         "cache.m5.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r5.2xlarge",
         "cache.r5.4xlarge",
         "cache.r5.12xlarge",
         "cache.r5.large",
         "cache.r5.xlarge"
     ]
        "ScaleDownModifications": [
         "cache.t2.micro",
         "cache.t2.small ",
         "cache.t2.medium "
     ],

 }
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Modify your existing cluster specifying the cluster to scale up and the new, larger node type, using the AWS CLI `modify-cache-cluster` command and the following parameters.

   - `--cache-cluster-id` – The name of the cluster you are scaling up.
   - `--cache-node-type` – The new node type you want to scale the cluster. This value must be one of the node types returned by the `list-allowed-node-type-modifications` command in step 1.
   - `--cache-parameter-group-name` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache

parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.

- `--apply-immediately` – Causes the scale-up process to be applied immediately. To postpone the scale-up process to the cluster's next maintenance window, use the `--no-apply-immediately` parameter.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-cluster \
     --cache-cluster-id my-redis-cache-cluster \
     --cache-node-type cache.m5.xlarge \
     --cache-parameter-group-name redis7-cluster \
     --apply-immediately
```

For Windows:

```
aws elasticache modify-cache-cluster ^
     --cache-cluster-id my-redis-cache-cluster ^
     --cache-node-type cache.m5.xlarge ^
     --cache-parameter-group-name redis7-cluster ^
     --apply-immediately
```

Output from the above command looks something like this (JSON format).

```
{
     "CacheCluster": {
         "Engine": "redis",
         "CacheParameterGroup": {
             "CacheNodeIdsToReboot": [],
             "CacheParameterGroupName": "default.redis6.x",
             "ParameterApplyStatus": "in-sync"
         },
         "SnapshotRetentionLimit": 1,
         "CacheClusterId": "my-redis-cache-cluster",
         "CacheSecurityGroups": [],
         "NumCacheNodes": 1,
         "SnapshotWindow": "00:00-01:00",
         "CacheClusterCreateTime": "2017-02-21T22:34:09.645Z",
         "AutoMinorVersionUpgrade": true,
         "CacheClusterStatus": "modifying",
```

```
         "PreferredAvailabilityZone": "us-west-2a",
         "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
         "CacheSubnetGroupName": "default",
         "EngineVersion": "6.0",
         "PendingModifiedValues": {
             "CacheNodeType": "cache.m5.2xlarge"
         },
         "PreferredMaintenanceWindow": "tue:11:30-tue:12:30",
         "CacheNodeType": "cache.m5.large",
          "DataTiering": "disabled"
     }
 }
```

For more information, see modify-cache-cluster in the *AWS CLI Reference*.

3. If you used the `--apply-immediately`, check the status of the new cluster using the AWS CLI `describe-cache-clusters` command with the following parameter. When the status changes to *available*, you can begin using the new, larger cluster.

   - `--cache-cluster-id` – The name of your single-node Valkey or Redis OSS cluster. Use this parameter to describe a particular cluster rather than all clusters.

```
aws elasticache describe-cache-clusters --cache-cluster-id my-redis-cache-cluster
```

For more information, see describe-cache-clusters in the *AWS CLI Reference*.

### Scaling up single-node Valkey or Redis OSS clusters (ElastiCache API)

The following procedure describes how to scale up a single-node Valkey or Redis OSS cluster using the ElastiCache API. During this process, your Valkey or Redis OSS cluster will continue to serve requests with minimal downtime.

### To scale up a single-node Valkey or Redis OSS cluster (ElastiCache API)

1. Determine the node types you can scale up to by running the ElastiCache API `ListAllowedNodeTypeModifications` action with the following parameter.

   - `CacheClusterId` – The name of the single-node Valkey or Redis OSS cluster you want to scale up.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ListAllowedNodeTypeModifications
    &CacheClusterId=MyRedisCacheCluster
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see ListAllowedNodeTypeModifications in the *Amazon ElastiCache API Reference*.

2. Modify your existing cluster specifying the cluster to scale up and the new, larger node type, using the `ModifyCacheCluster` ElastiCache API action and the following parameters.

   - `CacheClusterId` – The name of the cluster you are scaling up.
   - `CacheNodeType` – The new, larger node type you want to scale the cluster up to. This value must be one of the node types returned by the `ListAllowedNodeTypeModifications` action in the previous step.
   - `CacheParameterGroupName` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
   - `ApplyImmediately` – Set to `true` to cause the scale-up process to be performed immediately. To postpone the scale-up process to the cluster's next maintenance window, use `ApplyImmediately=false`.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyCacheCluster
    &ApplyImmediately=true
    &CacheClusterId=MyRedisCacheCluster
    &CacheNodeType=cache.m5.xlarge
    &CacheParameterGroupName redis7-cluster
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
```

```
    &X-Amz-Credential=<credential>
```

For more information, see ModifyCacheCluster in the *Amazon ElastiCache API Reference*.

3. If you used `ApplyImmediately=true`, check the status of the new cluster using the ElastiCache API `DescribeCacheClusters` action with the following parameter. When the status changes to *available*, you can begin using the new, larger cluster.

   - `CacheClusterId` – The name of your single-node Valkey or Redis OSS cluster. Use this parameter to describe a particular cluster rather than all clusters.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeCacheClusters
    &CacheClusterId=MyRedisCacheCluster
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see DescribeCacheClusters in the *Amazon ElastiCache API Reference*.

## Scaling down single-node Valkey or Redis OSS clusters

The following sections walk you through how to scale a single-node Valkey or Redis OSS cluster down to a smaller node type. Ensuring that the new, smaller node type is large enough to accommodate all the data and Valkey or Redis OSS overhead is important to the long-term success of your new Valkey or Redis OSS cluster. For more information, see Ensuring you have enough memory to make a Valkey or Redis OSS snapshot.

> **Note**
>
> For clusters running a data tiering node type, you can only scale to node sizes within the same data tiering node family.

### Topics

- Scaling down a single-node Valkey or Redis OSS cluster (Console)
- Scaling down single-node Valkey or Redis OSS clusters (AWS CLI)
- Scaling down single-node Valkey or Redis OSS clusters (ElastiCache API)

### Scaling down a single-node Valkey or Redis OSS cluster (Console)

The following procedure walks you through scaling your single-node Valkey or Redis OSS cluster down to a smaller node type using the ElastiCache console.

> **Important**
>
> If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, before you begin scaling be sure that you have a custom parameter group that reserves the correct amount of memory for your new node type. Alternatively, you can modify a custom parameter group so that it uses `reserved-memory-percent` and use that parameter group for your new cluster.
> If you're using `reserved-memory-percent`, doing this is not necessary.
> For more information, see Managing reserved memory for Valkey and Redis OSS.

#### To scale down your single-node Valkey or Redis OSS cluster (console)

1. Ensure that the smaller node type is adequate for your data and overhead needs.

2. If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, ensure that you have a custom parameter group to set aside the correct amount of memory for your new node type.

   Alternatively, you can modify your custom parameter group to use `reserved-memory-percent`. For more information, see Managing reserved memory for Valkey and Redis OSS.

3. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.

4. From the list of clusters, choose the cluster you want to scale down. This cluster must be running the Valkey or Redis OSS engine and not the clustered Valkey or Redis OSS engine.

5. Choose **Modify**.

6. In the **Modify Cluster** wizard:

   a. Choose the node type you want to scale down to from the **Node type** list.

   b. If you're using `reserved-memory` to manage your memory, from the **Parameter Group** list, choose the custom parameter group that reserves the correct amount of memory for your new node type.

7. If you want to perform the scale-down process right away, choose the **Apply immediately** check box. If the **Apply immediately** check box is left not chosen, the scale-down process is performed during this cluster's next maintenance window.

8. Choose **Modify**.

9. When the cluster's status changes from *modifying* to *available*, your cluster has scaled to the new node type. There is no need to update the endpoints in your application.

### Scaling down single-node Valkey or Redis OSS clusters (AWS CLI)

The following procedure describes how to scale down a single-node Valkey or Redis OSS cluster using the AWS CLI.

### To scale down a single-node Valkey or Redis OSS cluster (AWS CLI)

1. Determine the node types you can scale down to by running the AWS CLI `list-allowed-node-type-modifications` command with the following parameter.

   - `--cache-cluster-id`

For Linux, macOS, or Unix:

```
aws elasticache list-allowed-node-type-modifications \
     --cache-cluster-id my-cache-cluster-id
```

For Windows:

```
aws elasticache list-allowed-node-type-modifications ^
     --cache-cluster-id my-cache-cluster-id
```

Output from the above command looks something like this (JSON format).

```
{
     "ScaleUpModifications": [
         "cache.m5.2xlarge",
         "cache.m5.large",
         "cache.m5.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r5.2xlarge",
         "cache.r5.4xlarge",
         "cache.r5.12xlarge",
         "cache.r5.large",
         "cache.r5.xlarge"
     ]
        "ScaleDownModifications": [
         "cache.t2.micro",
         "cache.t2.small ",
         "cache.t2.medium "
     ],

 }
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Modify your existing cluster specifying the cluster to scale down and the new, smaller node type, using the AWS CLI `modify-cache-cluster` command and the following parameters.

- `--cache-cluster-id` – The name of the cluster you are scaling down.
- `--cache-node-type` – The new node type you want to scale the cluster. This value must be one of the node types returned by the `list-allowed-node-type-modifications` command in step 1.
- `--cache-parameter-group-name` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
- `--apply-immediately` – Causes the scale-down process to be applied immediately. To postpone the scale-up process to the cluster's next maintenance window, use the `--no-apply-immediately` parameter.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-cluster \
     --cache-cluster-id my-redis-cache-cluster \
     --cache-node-type cache.m5.xlarge \
     --cache-parameter-group-name redis7-cluster \
     --apply-immediately
```

For Windows:

```
aws elasticache modify-cache-cluster ^
     --cache-cluster-id my-redis-cache-cluster ^
     --cache-node-type cache.m5.xlarge ^
     --cache-parameter-group-name redis7-cluster ^
     --apply-immediately
```

Output from the above command looks something like this (JSON format).

```
{
     "CacheCluster": {
         "Engine": "redis",
         "CacheParameterGroup": {
             "CacheNodeIdsToReboot": [],
             "CacheParameterGroupName": "default.redis6,x",
             "ParameterApplyStatus": "in-sync"
```

```
        },
        "SnapshotRetentionLimit": 1,
        "CacheClusterId": "my-redis-cache-cluster",
        "CacheSecurityGroups": [],
        "NumCacheNodes": 1,
        "SnapshotWindow": "00:00-01:00",
        "CacheClusterCreateTime": "2017-02-21T22:34:09.645Z",
        "AutoMinorVersionUpgrade": true,
        "CacheClusterStatus": "modifying",
        "PreferredAvailabilityZone": "us-west-2a",
        "ClientDownloadLandingPage": "https://console.aws.amazon.com/elasticache/
home#client-download:",
        "CacheSubnetGroupName": "default",
        "EngineVersion": "6.0",
        "PendingModifiedValues": {
            "CacheNodeType": "cache.m5.2xlarge"
        },
        "PreferredMaintenanceWindow": "tue:11:30-tue:12:30",
        "CacheNodeType": "cache.m5.large",
         "DataTiering": "disabled"
    }
}
```

For more information, see modify-cache-cluster in the *AWS CLI Reference*.

3. If you used the `--apply-immediately`, check the status of the new cluster using the AWS CLI `describe-cache-clusters` command with the following parameter. When the status changes to *available*, you can begin using the new, smaller cluster.

   - `--cache-cluster-id` – The name of your single-node Valkey or Redis OSS cluster. Use this parameter to describe a particular cluster rather than all clusters.

```
aws elasticache describe-cache-clusters --cache-cluster-id my-redis-cache-cluster
```

For more information, see describe-cache-clusters in the *AWS CLI Reference*.

### Scaling down single-node Valkey or Redis OSS clusters (ElastiCache API)

The following procedure describes how to scale updown a single-node Valkey or Redis OSS cluster using the ElastiCache API.

**To scale down a single-node Valkey or Redis OSS cluster (ElastiCache API)**

1. Determine the node types you can scale down to by running the ElastiCache API `ListAllowedNodeTypeModifications` action with the following parameter.

   - `CacheClusterId` – The name of the single-node Valkey or Redis OSS cluster you want to scale down.

   ```
   https://elasticache.us-west-2.amazonaws.com/
       ?Action=ListAllowedNodeTypeModifications
       &CacheClusterId=MyRedisCacheCluster
       &Version=2015-02-02
       &SignatureVersion=4
       &SignatureMethod=HmacSHA256
       &Timestamp=20150202T192317Z
       &X-Amz-Credential=<credential>
   ```

   For more information, see ListAllowedNodeTypeModifications in the *Amazon ElastiCache API Reference*.

2. Modify your existing cluster specifying the cluster to scale up and the new, larger node type, using the `ModifyCacheCluster` ElastiCache API action and the following parameters.

   - `CacheClusterId` – The name of the cluster you are scaling down.
   - `CacheNodeType` – The new, smaller node type you want to scale the cluster down to. This value must be one of the node types returned by the `ListAllowedNodeTypeModifications` action in previous step.
   - `CacheParameterGroupName` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
   - `ApplyImmediately` – Set to `true` to cause the scale-down process to be performed immediately. To postpone the scale-up process to the cluster's next maintenance window, use `ApplyImmediately=false`.

   ```
   https://elasticache.us-west-2.amazonaws.com/
       ?Action=ModifyCacheCluster
       &ApplyImmediately=true
   ```

```
    &CacheClusterId=MyRedisCacheCluster
    &CacheNodeType=cache.m5.xlarge
    &CacheParameterGroupName redis7-cluster
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see ModifyCacheCluster in the *Amazon ElastiCache API Reference*.

3. If you used `ApplyImmediately=true`, check the status of the new cluster using the ElastiCache API `DescribeCacheClusters` action with the following parameter. When the status changes to *available*, you can begin using the new, smaller cluster.

   - `CacheClusterId` – The name of your single-node Valkey or Redis OSS cluster. Use this parameter to describe a particular cluster rather than all clusters.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeCacheClusters
    &CacheClusterId=MyRedisCacheCluster
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see DescribeCacheClusters in the *Amazon ElastiCache API Reference*.

## Scaling replica nodes for Valkey or Redis OSS (Cluster Mode Disabled)

A Valkey or Redis OSS cluster with replica nodes (called *replication group* in the API/CLI) provides high availability via replication that has Multi-AZ with automatic failover enabled. A cluster with replica nodes is a logical collection of up to six Valkey or Redis OSS nodes where one node, the Primary, is able to serve both read and write requests. All the other nodes in the cluster are read-only replicas of the Primary. Data written to the Primary is asynchronously replicated to all the read replicas in the cluster. Because Valkey or Redis OSS (cluster mode disabled) does not support partitioning your data across multiple clusters, each node in a Valkey or Redis OSS (cluster mode disabled) replication group contains the entire cache dataset. Valkey or Redis OSS (cluster mode enabled) clusters support partitioning your data across up to 500 shards.

To change the data capacity of your cluster you must scale it up to a larger node type, or down to a smaller node type.

To change the read capacity of your cluster, add more read replicas, up to a maximum of 5, or remove read replicas.

The ElastiCache scaling up process is designed to make a best effort to retain your existing data and requires successful Valkey or Redis OSS replication. For Valkey or Redis OSS clusters with replicas, we recommend that sufficient memory be made available to Valkey or Redis OSS.

**Topics**


- Scaling up Valkey or Redis OSS clusters with replicas
- Scaling down Valkey or Redis OSS clusters with replicas
- Increasing read capacity
- Decreasing read capacity


**Related Topics**

- High availability using replication groups
- Replication: Valkey and Redis OSS Cluster Mode Disabled vs. Enabled
- Minimizing downtime in ElastiCache by using Multi-AZ with Valkey and Redis OSS
- Ensuring you have enough memory to make a Valkey or Redis OSS snapshot

**Topics**

- Scaling up Valkey or Redis OSS clusters with replicas
- Scaling down Valkey or Redis OSS clusters with replicas
- Increasing read capacity
- Decreasing read capacity

### Scaling up Valkey or Redis OSS clusters with replicas

Amazon ElastiCache provides console, CLI, and API support for scaling your Valkey or Redis OSS (cluster mode disabled) replication group up.

When the scale-up process is initiated, ElastiCache does the following:

1. Launches a replication group using the new node type.
2. Copies all the data from the current primary node to the new primary node.
3. Syncs the new read replicas with the new primary node.
4. Updates the DNS entries so they point to the new nodes. Because of this you don't have to update the endpoints in your application. For Valkey 7.2 and above or Redis OSS 5.0.5 and above, you can scale auto failover enabled clusters while the cluster continues to stay online and serve incoming requests. On Redis OSS version 4.0.10 and below, you may notice a brief interruption of reads and writes on previous versions from the primary node while the DNS entry is updated.
5. Deletes the old nodes (CLI/API: replication group). You will notice a brief interruption (a few seconds) of reads and writes from the old nodes because the connections to the old nodes will be disconnected.

How long this process takes is dependent upon your node type and how much data is in your cluster.

As shown in the following table, your Valkey or Redis OSS scale-up operation is blocked if you have an engine upgrade scheduled for the cluster's next maintenance window.

**Blocked Valkey or Redis OSS operations**

| Pending Operations | Blocked Operations |
|---|---|
| Scale up | Immediate engine upgrade |
| Engine upgrade | Immediate scale up |
| Scale up and engine upgrade | Immediate scale up |
| | Immediate engine upgrade |

If you have a pending operation that is blocking you, you can do one of the following.

- Schedule your Valkey or Redis OSS scale-up operation for the next maintenance window by clearing the **Apply immediately** check box (CLI use: `--no-apply-immediately`, API use: `ApplyImmediately=false`).
- Wait until your next maintenance window (or after) to perform your Valkey or Redis OSS scale-up operation.
- Add the Valkey or Redis OSS engine upgrade to this cluster modification with the **Apply Immediately** check box chosen (CLI use: `--apply-immediately`, API use: `ApplyImmediately=true`). This unblocks your scale-up operation by causing the engine upgrade to be performed immediately.

The following sections describe how to scale your Valkey or Redis OSS cluster with replicas up using the ElastiCache console, the AWS CLI, and the ElastiCache API.

> **Important**
>
> If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, before you begin scaling be sure that you have a custom parameter group that reserves the correct amount of memory for your new node type. Alternatively, you can modify a custom parameter group so that it uses `reserved-memory-percent` and use that parameter group for your new cluster.
> If you're using `reserved-memory-percent`, doing this is not necessary.
> For more information, see Managing reserved memory for Valkey and Redis OSS.

### Scaling up a Valkey or Redis OSS cluster with replicas (Console)

The amount of time it takes to scale up to a larger node type varies, depending upon the node type and the amount of data in your current cluster.

The following process scales your cluster with replicas from its current node type to a new, larger node type using the ElastiCache console. During this process, there may be a brief interruption of reads and writes for other versions from the primary node while the DNS entry is updated. you might see less than 1 second downtime for nodes running on 5.0.6 versions and above and a few seconds for older versions.

**To scale up Valkey or Redis OSS cluster with replicas (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**
3. From the list of clusters, choose the cluster you want to scale up. This cluster must be running the Valkey or Redis OSS engine and not the clustered Valkey or Redis OSS engine.
4. Choose **Modify**.
5. In the **Modify Cluster** wizard:
    a. Choose the node type you want to scale to from the **Node type** list. Note that not all node types are available to scale down to.
    b. If you're using `reserved-memory` to manage your memory, from the **Parameter Group** list, choose the custom parameter group that reserves the correct amount of memory for your new node type.
6. If you want to perform the scale-up process right away, choose the **Apply immediately** check box. If the **Apply immediately** check box is left not chosen, the scale-up process is performed during this cluster's next maintenance window.
7. Choose **Modify**.
8. When the cluster's status changes from *modifying* to *available*, your cluster has scaled to the new node type. There is no need to update the endpoints in your application.

**Scaling up a Valkey or Redis OSS replication group (AWS CLI)**

The following process scales your replication group from its current node type to a new, larger node type using the AWS CLI. During this process, ElastiCache updates the DNS entries so they point to the new nodes. Because of this you don't have to update the endpoints in your application. For Valkey 7.2 and above or Redis OSS 5.0.5 and above, you can scale auto failover enabled clusters while the cluster continues to stay online and serve incoming requests. On version 4.0.10 and below, you may notice a brief interruption of reads and writes on previous versions from the primary node while the DNS entry is updated..

The amount of time it takes to scale up to a larger node type varies, depending upon your node type and the amount of data in your current cluster.

**To scale up a Valkey or Redis OSS Replication Group (AWS CLI)**

1. Determine which node types you can scale up to by running the AWS CLI `list-allowed-node-type-modifications` command with the following parameter.

   - `--replication-group-id` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

   For Linux, macOS, or Unix:

   ```
   aws elasticache list-allowed-node-type-modifications \
        --replication-group-id my-repl-group
   ```

   For Windows:

   ```
   aws elasticache list-allowed-node-type-modifications ^
        --replication-group-id my-repl-group
   ```

   Output from this operation looks something like this (JSON format).

   ```
   {
        "ScaleUpModifications": [
            "cache.m5.2xlarge",
            "cache.m5.large",
            "cache.m5.xlarge",
            "cache.m4.10xlarge",
            "cache.m4.2xlarge",
            "cache.m4.4xlarge",
            "cache.m4.large",
            "cache.m4.xlarge",
            "cache.r5.2xlarge",
            "cache.r5.4xlarge",
            "cache.r5.12xlarge",
            "cache.r5.large",
            "cache.r5.xlarge"
        ]
    }
   ```

   For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Scale your current replication group up to the new node type using the AWS CLI `modify-replication-group` command with the following parameters.

   - `--replication-group-id` – the name of the replication group.
   - `--cache-node-type` – the new, larger node type of the clusters in this replication group. This value must be one of the instance types returned by the `list-allowed-node-type-modifications` command in the previous step.
   - `--cache-parameter-group-name` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
   - `--apply-immediately` – Causes the scale-up process to be applied immediately. To postpone the scale-up operation to the next maintenance window, use `--no-apply-immediately`.

   For Linux, macOS, or Unix:

   ```
   aws elasticache modify-replication-group \
        --replication-group-id my-repl-group \
        --cache-node-type cache.m5.xlarge \
        --cache-parameter-group-name redis7-cluster \
        --apply-immediately
   ```

   For Windows:

   ```
   aws elasticache modify-replication-group ^
        --replication-group-id my-repl-group ^
        --cache-node-type cache.m5.xlarge ^
        --cache-parameter-group-name redis7-cluster \
        --apply-immediately
   ```

   Output from this command looks something like this (JSON format).

   ```
   {
    "ReplicationGroup": {
     "Status": "available",
     "Description": "Some description",
   ```

```
  "NodeGroups": [{
   "Status": "available",
   "NodeGroupMembers": [{
     "CurrentRole": "primary",
     "PreferredAvailabilityZone": "us-west-2b",
     "CacheNodeId": "0001",
     "ReadEndpoint": {
      "Port": 6379,
      "Address": "my-repl-group-001.8fdx4s.0001.usw2.cache.amazonaws.com"
     },
     "CacheClusterId": "my-repl-group-001"
    },
    {
     "CurrentRole": "replica",
     "PreferredAvailabilityZone": "us-west-2c",
     "CacheNodeId": "0001",
     "ReadEndpoint": {
      "Port": 6379,
      "Address": "my-repl-group-002.8fdx4s.0001.usw2.cache.amazonaws.com"
     },
     "CacheClusterId": "my-repl-group-002"
    }
   ],
   "NodeGroupId": "0001",
   "PrimaryEndpoint": {
    "Port": 6379,
    "Address": "my-repl-group.8fdx4s.ng.0001.usw2.cache.amazonaws.com"
   }
  }],
  "ReplicationGroupId": "my-repl-group",
  "SnapshotRetentionLimit": 1,
  "AutomaticFailover": "disabled",
  "SnapshotWindow": "12:00-13:00",
  "SnapshottingClusterId": "my-repl-group-002",
  "MemberClusters": [
   "my-repl-group-001",
   "my-repl-group-002"
  ],
  "PendingModifiedValues": {}
 }
}
```

For more information, see modify-replication-group in the *AWS CLI Reference*.

3. If you used the `--apply-immediately` parameter, monitor the status of the replication group using the AWS CLI `describe-replication-group` command with the following parameter. While the status is still in *modifying*, you might see less than 1 second downtime for nodes running on 5.0.6 versions and above and a brief interruption of reads and writes for older versions from the primary node while the DNS entry is updated.

   - `--replication-group-id` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

   For Linux, macOS, or Unix:

   ```
   aws elasticache describe-replication-groups \
        --replication-group-id my-replication-group
   ```

   For Windows:

   ```
   aws elasticache describe-replication-groups ^
        --replication-group-id my-replication-group
   ```

   For more information, see describe-replication-groups in the *AWS CLI Reference*.

**Scaling up a Valkey or Redis OSS replication group (ElastiCache API)**

The following process scales your replication group from its current node type to a new, larger node type using the ElastiCache API. For Valkey 7.2 and above or Redis OSS 5.0.5 and above, you can scale auto failover enabled clusters while the cluster continues to stay online and serve incoming requests. On version Redis OSS 4.0.10 and below, you may notice a brief interruption of reads and writes on previous versions from the primary node while the DNS entry is updated.

The amount of time it takes to scale up to a larger node type varies, depending upon your node type and the amount of data in your current cluster.

**To scale up a Valkey or Redis OSS Replication Group (ElastiCache API)**

1. Determine which node types you can scale up to using the ElastiCache API `ListAllowedNodeTypeModifications` action with the following parameter.

- `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a specific replication group rather than all replication groups.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ListAllowedNodeTypeModifications
    &ReplicationGroupId=MyReplGroup
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see ListAllowedNodeTypeModifications in the *Amazon ElastiCache API Reference*.

2. Scale your current replication group up to the new node type using the `ModifyReplicationGroup` ElastiCache API action and with the following parameters.

- `ReplicationGroupId` – the name of the replication group.
- `CacheNodeType` – the new, larger node type of the clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in th previous step.
- `CacheParameterGroupName` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
- `ApplyImmediately` – Set to `true` to causes the scale-up process to be applied immediately. To postpone the scale-up process to the next maintenance window, use `ApplyImmediately=false`.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyReplicationGroup
    &ApplyImmediately=true
    &CacheNodeType=cache.m5.2xlarge
    &CacheParameterGroupName=redis7-cluster
    &ReplicationGroupId=myReplGroup
    &SignatureVersion=4
```

```
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &Version=2014-12-01
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ModifyReplicationGroup in the *Amazon ElastiCache API Reference*.

3. If you used `ApplyImmediately=true`, monitor the status of the replication group using the ElastiCache API `DescribeReplicationGroups` action with the following parameters. When the status changes from *modifying* to *available*, you can begin writing to your new, scaled up replication group.

   - `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeReplicationGroups
    &ReplicationGroupId=MyReplGroup
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see DescribeReplicationGroups in the *Amazon ElastiCache API Reference*.

### Scaling down Valkey or Redis OSS clusters with replicas

The following sections walk you through how to scale a Valkey or Redis OSS (cluster mode disabled) cluster with replica nodes down to a smaller node type. Ensuring that the new, smaller node type is large enough to accommodate all the data and overhead is very important to success. For more information, see Ensuring you have enough memory to make a Valkey or Redis OSS snapshot.

> **Note**
>
> For clusters running a data tiering node type, you can only scale to node sizes within the same data tiering node family.

> **Important**
>
> If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, before you begin scaling be sure that you have a custom parameter group that reserves the correct amount of memory for your new node type. Alternatively, you can modify a custom parameter group so that it uses `reserved-memory-percent` and use that parameter group for your new cluster.
> If you're using `reserved-memory-percent`, doing this is not necessary.
> For more information, see Managing reserved memory for Valkey and Redis OSS.

#### Scaling down a Valkey or Redis OSS cluster with replicas (Console)

The following process scales your Valkey or Redis OSS cluster with replica nodes to a smaller node type using the ElastiCache console.

**To scale down a Valkey or Redis OSS cluster with replica nodes (console)**

1. Ensure that the smaller node type is adequate for your data and overhead needs.
2. If your parameter group uses `reserved-memory` to set aside memory for Valkey or Redis OSS overhead, ensure that you have a custom parameter group to set aside the correct amount of memory for your new node type.

   Alternatively, you can modify your custom parameter group to use `reserved-memory-percent`. For more information, see Managing reserved memory for Valkey and Redis OSS.

3. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
4. From the list of clusters, choose the cluster you want to scale down. This cluster must be running the Valkey or Redis OSS engine and not the clustered Valkey or Redis OSS engine.
5. Choose **Modify**.
6. In the **Modify Cluster** wizard:
   a. Choose the node type you want to scale down to from the **Node type** list.
   b. If you're using `reserved-memory` to manage your memory, from the **Parameter Group** list, choose the custom parameter group that reserves the correct amount of memory for your new node type.
7. If you want to perform the scale-down process right away, choose the **Apply immediately** check box. If the **Apply immediately** check box is left not chosen, the scale-down process is performed during this cluster's next maintenance window.
8. Choose **Modify**.
9. When the cluster's status changes from *modifying* to *available*, your cluster has scaled to the new node type. There is no need to update the endpoints in your application.

### Scaling down a Valkey or Redis OSS replication group (AWS CLI)

The following process scales your replication group from its current node type to a new, smaller node type using the AWS CLI. During this process, ElastiCache updates the DNS entries so they point to the new nodes. Because of this you don't have to update the endpoints in your application. For Valkey 7.2 above or Redis OSS 5.0.5 and above, you can scale auto failover enabled clusters while the cluster continues to stay online and serve incoming requests. On version 4.0.10 and below, you may notice a brief interruption of reads and writes on previous versions from the primary node while the DNS entry is updated..

However, reads from the read replica clusters continue uninterrupted.

The amount of time it takes to scale down to a smaller node type varies, depending upon your node type and the amount of data in your current cluster.

### To scale down a Valkey or Redis OSS Replication Group (AWS CLI)

1. Determine which node types you can scale down to by running the AWS CLI `list-allowed-node-type-modifications` command with the following parameter.

- `--replication-group-id` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

For Linux, macOS, or Unix:

```
aws elasticache list-allowed-node-type-modifications \
     --replication-group-id my-repl-group
```

For Windows:

```
aws elasticache list-allowed-node-type-modifications ^
     --replication-group-id my-repl-group
```

Output from this operation looks something like this (JSON format).

```
{
     "ScaleDownModifications": [
         "cache.m5.2xlarge",
         "cache.m5.large",
         "cache.m5.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r5.2xlarge",
         "cache.r5.4xlarge",
         "cache.r5.12xlarge",
         "cache.r5.large",
         "cache.r5.xlarge"
     ]
 }
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Scale your current replication group up to the new node type using the AWS CLI `modify-replication-group` command with the following parameters.

- `--replication-group-id` – the name of the replication group.

- `--cache-node-type` – the new, smaller node type of the clusters in this replication group. This value must be one of the instance types returned by the `list-allowed-node-type-modifications` command in the previous step.
- `--cache-parameter-group-name` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
- `--apply-immediately` – Causes the scale-up process to be applied immediately. To postpone the scale-up operation to the next maintenance window, use `--no-apply-immediately`.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
     --replication-group-id my-repl-group \
     --cache-node-type cache.t2.small  \
     --cache-parameter-group-name redis7-cluster \
     --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
     --replication-group-id my-repl-group ^
     --cache-node-type cache.t2.small  ^
     --cache-parameter-group-name redis7-cluster \
     --apply-immediately
```

Output from this command looks something like this (JSON format).

```
{"ReplicationGroup": {
         "Status": "available",
         "Description": "Some description",
         "NodeGroups": [
             {
                 "Status": "available",
                 "NodeGroupMembers": [
                     {
```

```
                        "CurrentRole": "primary",
                        "PreferredAvailabilityZone": "us-west-2b",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address": "my-repl-
group-001.8fdx4s.0001.usw2.cache.amazonaws.com"
                        },
                        "CacheClusterId": "my-repl-group-001"
                    },
                    {
                        "CurrentRole": "replica",
                        "PreferredAvailabilityZone": "us-west-2c",
                        "CacheNodeId": "0001",
                        "ReadEndpoint": {
                            "Port": 6379,
                            "Address": "my-repl-
group-002.8fdx4s.0001.usw2.cache.amazonaws.com"
                        },
                        "CacheClusterId": "my-repl-group-002"
                    }
                ],
                "NodeGroupId": "0001",
                "PrimaryEndpoint": {
                    "Port": 6379,
                    "Address": "my-repl-
group.8fdx4s.ng.0001.usw2.cache.amazonaws.com"
                }
            }
        ],
        "ReplicationGroupId": "my-repl-group",
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "disabled",
        "SnapshotWindow": "12:00-13:00",
        "SnapshottingClusterId": "my-repl-group-002",
        "MemberClusters": [
            "my-repl-group-001",
            "my-repl-group-002",
        ],
        "PendingModifiedValues": {}
    }
 }
```

For more information, see modify-replication-group in the *AWS CLI Reference*.

3. If you used the `--apply-immediately` parameter, monitor the status of the replication group using the AWS CLI `describe-replication-group` command with the following parameter. When the status changes from *modifying* to *available*, you can begin writing to your new, scaled down replication group.

   - `--replication-group-id` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

   For Linux, macOS, or Unix:

   ```
   aws elasticache describe-replication-group \
        --replication-group-id my-replication-group
   ```

   For Windows:

   ```
   aws elasticache describe-replication-groups ^
        --replication-group-id my-replication-group
   ```

   For more information, see describe-replication-groups in the *AWS CLI Reference*.

**Scaling down a Valkey or Redis OSS replication group (ElastiCache API)**

The following process scales your replication group from its current node type to a new, smaller node type using the ElastiCache API. During this process, ElastiCache updates the DNS entries so they point to the new nodes. Because of this you don't have to update the endpoints in your application. For Valkey 7.2 and above or Redis OSS 5.0.5 and above, you can scale auto failover enabled clusters while the cluster continues to stay online and serve incoming requests. On Redis OSS version 4.0.10 and below, you may notice a brief interruption of reads and writes on previous versions from the primary node while the DNS entry is updated.. However, reads from the read replica clusters continue uninterrupted.

The amount of time it takes to scale down to a smaller node type varies, depending upon your node type and the amount of data in your current cluster.

**To scale down a Valkey or Redis OSS Replication Group (ElastiCache API)**

1. Determine which node types you can scale down to using the ElastiCache API `ListAllowedNodeTypeModifications` action with the following parameter.

   - `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a specific replication group rather than all replication groups.

   ```
   https://elasticache.us-west-2.amazonaws.com/
       ?Action=ListAllowedNodeTypeModifications
       &ReplicationGroupId=MyReplGroup
       &Version=2015-02-02
       &SignatureVersion=4
       &SignatureMethod=HmacSHA256
       &Timestamp=20150202T192317Z
       &X-Amz-Credential=<credential>
   ```

   For more information, see ListAllowedNodeTypeModifications in the *Amazon ElastiCache API Reference*.

2. Scale your current replication group up to the new node type using the `ModifyReplicationGroup` ElastiCache API action and with the following parameters.

   - `ReplicationGroupId` – the name of the replication group.
   - `CacheNodeType` – the new, smaller node type of the clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in the previous step.
   - `CacheParameterGroupName` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
   - `ApplyImmediately` – Set to `true` to causes the scale-up process to be applied immediately. To postpone the scale-down process to the next maintenance window, use `ApplyImmediately=false`.

   ```
   https://elasticache.us-west-2.amazonaws.com/
       ?Action=ModifyReplicationGroup
       &ApplyImmediately=true
   ```

```
    &CacheNodeType=cache.m5.2xlarge
    &CacheParameterGroupName=redis7-cluster
    &ReplicationGroupId=myReplGroup
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &Version=2014-12-01
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ModifyReplicationGroup in the *Amazon ElastiCache API Reference*.

3. If you used `ApplyImmediately=true`, monitor the status of the replication group using the ElastiCache API `DescribeReplicationGroups` action with the following parameters. When the status changes from *modifying* to *available*, you can begin writing to your new, scaled down replication group.

   - `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeReplicationGroups
    &ReplicationGroupId=MyReplGroup
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see DescribeReplicationGroups in the *Amazon ElastiCache API Reference*.

### Increasing read capacity

To increase read capacity, add read replicas (up to a maximum of five) to your Valkey or Redis OSS replication group.

You can scale your Valkey or Redis OSS cluster's read capacity using the ElastiCache console, the AWS CLI, or the ElastiCache API. For more information, see Adding a read replica for Valkey or Redis OSS (Cluster Mode Disabled).

### Decreasing read capacity

To decrease read capacity, delete one or more read replicas from your Valkey or Redis OSS cluster with replicas (called *replication group* in the API/CLI). If the cluster is Multi-AZ with automatic failover enabled, you cannot delete the last read replica without first disabling Multi-AZ. For more information, see Modifying a replication group.

For more information, see Deleting a read replica for Valkey or Redis OSS (Cluster Mode Disabled).

## Scaling Valkey or Redis OSS (Cluster Mode Enabled) clusters

As demand on your clusters changes, you might decide to improve performance or reduce costs by changing the number of shards in your Valkey or Redis OSS (cluster mode enabled) cluster. We recommend using online horizontal scaling to do so, because it allows your cluster to continue serving requests during the scaling process.

Conditions under which you might decide to rescale your cluster include the following:

- **Memory pressure:**

  If the nodes in your cluster are under memory pressure, you might decide to scale out so that you have more resources to better store data and serve requests.

  You can determine whether your nodes are under memory pressure by monitoring the following metrics: *FreeableMemory*, *SwapUsage*, and *BytesUsedForCache*.

- **CPU or network bottleneck:**

  If latency/throughput issues are plaguing your cluster, you might need to scale out to resolve the issues.

  You can monitor your latency and throughput levels by monitoring the following metrics: *CPUUtilization*, *NetworkBytesIn*, *NetworkBytesOut*, *CurrConnections*, and *NewConnections*.

- **Your cluster is over-scaled:**

  Current demand on your cluster is such that scaling in doesn't hurt performance and reduces your costs.

  You can monitor your cluster's use to determine whether or not you can safely scale in using the following metrics: *FreeableMemory*, *SwapUsage*, *BytesUsedForCache*, *CPUUtilization*, *NetworkBytesIn*, *NetworkBytesOut*, *CurrConnections*, and *NewConnections*.

- **Durability write throughput limit:**

  Durability-enabled clusters support up to 100 MBps of write throughput per primary node. If your workload approaches this limit, you should scale out by adding shards to distribute write load across more primary nodes.

  You can detect when throughput saturation is approaching by monitoring the following metrics: *MaxReplicationThroughput*, *DurabilityLag*, and *TrafficManagementActive*.

### Performance Impact of Scaling

When you scale using the offline process, your cluster is offline for a significant portion of the process and thus unable to serve requests. When you scale using the online method, because scaling is a compute-intensive operation, there is some degradation in performance, nevertheless, your cluster continues to serve requests throughout the scaling operation. How much degradation you experience depends upon your normal CPU utilization and your data.

There are two ways to scale your Valkey or Redis OSS (cluster mode enabled) cluster; horizontal and vertical scaling.

- Horizontal scaling allows you to change the number of node groups (shards) in the replication group by adding or removing node groups (shards). The online resharding process allows scaling in/out while the cluster continues serving incoming requests.

  Configure the slots in your new cluster differently than they were in the old cluster. Offline method only.
- Vertical Scaling - Change the node type to resize the cluster. The online vertical scaling allows scaling up/down while the cluster continues serving incoming requests.

If you are reducing the size and memory capacity of the cluster, by either scaling in or scaling down, ensure that the new configuration has sufficient memory for your data and Valkey or Redis OSS overhead.

For more information, see Choosing your node size.

**Contents**


- Offline resharding for Valkey or Redis OSS (cluster mode enabled)
- Online resharding for Valkey or Redis OSS (cluster mode enabled)
  - Adding shards with online resharding
  - Removing shards with online resharding
    - Removing shards (Console)
    - Removing shards (AWS CLI)
    - Removing shards (ElastiCache API)
  - Online shard rebalancing
    - Online Shard Rebalancing (Console)

    - Online shard rebalancing (AWS CLI)
    - Online shard rebalancing (ElastiCache API)
- Online vertical scaling by modifying node type
  - Online scaling up
    - Scaling up Valkey or Redis OSS clusters (Console)
    - Scaling up Valkey or Redis OSS clusters (AWS CLI)
    - Scaling up Valkey or Redis OSS clusters (ElastiCache API)
  - Online scaling down
    - Scaling down Valkey or Redis OSS clusters (Console)
    - Scaling down Valkey or Redis OSS clusters (AWS CLI)
    - Scaling down Valkey or Redis OSS clusters (ElastiCache API)

**Offline resharding for Valkey or Redis OSS (cluster mode enabled)**

The main advantage you get from offline shard reconfiguration is that you can do more than merely add or remove shards from your replication group. When you reshard and rebalance offline, in addition to changing the number of shards in your replication group, you can do the following:

> **Note**
>
> Offline resharding is not supported on Valkey or Redis OSS clusters with data tiering enabled. For more information, see Data tiering in ElastiCache.

- Change the node type of your replication group.
- Specify the Availability Zone for each node in the replication group.
- Upgrade to a newer engine version.
- Specify the number of replica nodes in each shard independently.
- Specify the keyspace for each shard.

The main disadvantage of offline shard reconfiguration is that your cluster is offline beginning with the restore portion of the process and continuing until you update the endpoints in your application. The length of time that your cluster is offline varies with the amount of data in your cluster.

**To reconfigure your shards Valkey or Redis OSS (cluster mode enabled) cluster offline**

1. Create a manual backup of your existing Valkey or Redis OSS cluster. For more information, see Taking manual backups.
2. Create a new cluster by restoring from the backup. For more information, see Restoring from a backup into a new cache.
3. Update the endpoints in your application to the new cluster's endpoints. For more information, see Finding connection endpoints in ElastiCache.

### Online resharding for Valkey or Redis OSS (cluster mode enabled)

By using online resharding and shard rebalancing with ElastiCache Valkey and Redis OSS, you can scale your Valkey or Redis OSS (cluster mode enabled) cluster dynamically with no downtime. This approach means that your cluster can continue to serve requests even while scaling or rebalancing is in process.

You can do the following:

- **Scale out** – Increase read and write capacity by adding shards (node groups) to your Valkey or Redis OSS (cluster mode enabled) cluster (replication group).

  If you add one or more shards to your replication group, the number of nodes in each new shard is the same as the number of nodes in the smallest of the existing shards.
- **Scale in** – Reduce read and write capacity, and thereby costs, by removing shards from your Valkey or Redis OSS (cluster mode enabled) cluster.
- **Rebalance** – Move the keyspaces among the shards in your Valkey or Redis OSS (cluster mode enabled) cluster so they are as equally distributed among the shards as possible.

You can't do the following:

- **Configure shards independently:**

  You can't specify the keyspace for shards independently. To do this, you must use the offline process.

Currently, the following limitations apply to ElastiCache online resharding and rebalancing:

- There are limitations with slots or keyspaces and large items:

If any of the keys in a shard contain a large item, that key isn't migrated to a new shard when scaling out or rebalancing. This functionality can result in unbalanced shards.

If any of the keys in a shard contain a large item (items greater than 256 MB after serialization), that shard isn't deleted when scaling in. This functionality can result in some shards not being deleted.

- When scaling out, the number of nodes in any new shards equals the number of nodes in the smallest existing shard.
- When scaling out, any tags that are common to all existing shards are copied to the new shards.
- When scaling out a Global Data Store cluster, ElastiCache will not automatically replicate Functions from one of the existing nodes to the new node(s). We recommend loading your Functions in the new shard(s) after scaling out your cluster so that every shards have the same functions.

> **Note**
>
> In ElastiCache for Valkey 7.2 and above, and ElastiCache for Redis OSS version 7 and above: When scaling out your cluster, ElastiCache will automatically replicate the Functions loaded in one of the existing nodes (selected at random) to the new node(s). If your application uses Functions, we recommend loading all of your functions to all the shards before scaling out so that your cluster does not end up with different function definitions on different shards.

For more information, see Online cluster resizing.

You can horizontally scale or rebalance your Valkey or Redis OSS (cluster mode enabled) clusters using the AWS Management Console, the AWS CLI, and the ElastiCache API.

**Adding shards with online resharding**

You can add shards to your Valkey or Redis OSS (cluster mode enabled) cluster using the AWS Management Console, AWS CLI, or ElastiCache API. When you add shards to a Valkey or Redis OSS (cluster mode enabled) cluster, any tags on the existing shards are copied over to the new shards.

### Adding shards (Console)

You can use the AWS Management Console to add one or more shards to your Valkey or Redis OSS (cluster mode enabled) cluster. The following procedure describes the process.

**To add shards to your Valkey or Redis OSS (cluster mode enabled) cluster**

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. Locate and choose the name, not the box to the left of the cluster's name, of the Valkey or Redis OSS (cluster mode enabled) cluster that you want to add shards to.

   > **Tip**
   >
   > Valkey or Redis OSS (cluster mode enabled) show **Clustered Valkey** or **Clustered Redis OSS** in the **Mode** column

4. Choose **Add shard**.

   a. For **Number of shards to be added**, choose the number of shards you want added to this cluster.

   b. For **Availability zone(s)**, choose either **No preference** or **Specify availability zones**.

   c. If you chose **Specify availability zones**, for each node in each shard, select the node's Availability Zone from the list of Availability Zones.

   d. Choose **Add**.

### Adding shards (AWS CLI)

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by adding shards using the AWS CLI.

Use the following parameters with `modify-replication-group-shard-configuration`.

**Parameters**

- `--apply-immediately` – Required. Specifies the shard reconfiguration operation is to be started immediately.
- `--replication-group-id` – Required. Specifies which replication group (cluster) the shard reconfiguration operation is to be performed on.

- `--node-group-count` – Required. Specifies the number of shards (node groups) to exist when the operation is completed. When adding shards, the value of `--node-group-count` must be greater than the current number of shards.

  Optionally, you can specify the Availability Zone for each node in the replication group using `--resharding-configuration`.
- `--resharding-configuration` – Optional. A list of preferred Availability Zones for each node in each shard in the replication group. Use this parameter only if the value of `--node-group-count` is greater than the current number of shards. If this parameter is omitted when adding shards, Amazon ElastiCache selects the Availability Zones for the new nodes.

The following example reconfigures the keyspaces over four shards in a Valkey or Redis OSS (cluster mode enabled) cluster named `my-cluster`. The example also specifies the Availability Zone for each node in each shard. The operation begins immediately.

**Example- Adding Shards**

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group-shard-configuration \
    --replication-group-id my-cluster \
    --node-group-count 4 \
    --resharding-configuration \
        "PreferredAvailabilityZones=us-east-2a,us-east-2c" \
        "PreferredAvailabilityZones=us-east-2b,us-east-2a" \
        "PreferredAvailabilityZones=us-east-2c,us-east-2d" \
        "PreferredAvailabilityZones=us-east-2d,us-east-2c" \
    --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group-shard-configuration ^
    --replication-group-id my-cluster ^
    --node-group-count 4 ^
    --resharding-configuration ^
        "PreferredAvailabilityZones=us-east-2a,us-east-2c" ^
        "PreferredAvailabilityZones=us-east-2b,us-east-2a" ^
        "PreferredAvailabilityZones=us-east-2c,us-east-2d" ^
        "PreferredAvailabilityZones=us-east-2d,us-east-2c" ^
    --apply-immediately
```

For more information, see modify-replication-group-shard-configuration in the AWS CLI documentation.

**Adding shards (ElastiCache API)**

You can use the ElastiCache API to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster online by using the `ModifyReplicationGroupShardConfiguration` operation.

Use the following parameters with `ModifyReplicationGroupShardConfiguration`.

**Parameters**

- `ApplyImmediately=true` – Required. Specifies the shard reconfiguration operation is to be started immediately.
- `ReplicationGroupId` – Required. Specifies which replication group (cluster) the shard reconfiguration operation is to be performed on.
- `NodeGroupCount` – Required. Specifies the number of shards (node groups) to exist when the operation is completed. When adding shards, the value of `NodeGroupCount` must be greater than the current number of shards.

  Optionally, you can specify the Availability Zone for each node in the replication group using `ReshardingConfiguration`.
- `ReshardingConfiguration` – Optional. A list of preferred Availability Zones for each node in each shard in the replication group. Use this parameter only if the value of `NodeGroupCount` is greater than the current number of shards. If this parameter is omitted when adding shards, Amazon ElastiCache selects the Availability Zones for the new nodes.

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by adding shards using the ElastiCache API.

**Example- Adding Shards**

The following example adds node groups to the Valkey or Redis OSS (cluster mode enabled) cluster `my-cluster`, so there are a total of four node groups when the operation completes. The example also specifies the Availability Zone for each node in each shard. The operation begins immediately.

```
https://elasticache.us-east-2.amazonaws.com/
```

```
    ?Action=ModifyReplicationGroupShardConfiguration
    &ApplyImmediately=true
    &NodeGroupCount=4
    &ReplicationGroupId=my-cluster

 &ReshardingConfiguration.ReshardingConfiguration.1.PreferredAvailabilityZones.AvailabilityZone
east-2a

 &ReshardingConfiguration.ReshardingConfiguration.1.PreferredAvailabilityZones.AvailabilityZone
east-2c

 &ReshardingConfiguration.ReshardingConfiguration.2.PreferredAvailabilityZones.AvailabilityZone
east-2b

 &ReshardingConfiguration.ReshardingConfiguration.2.PreferredAvailabilityZones.AvailabilityZone
east-2a

 &ReshardingConfiguration.ReshardingConfiguration.3.PreferredAvailabilityZones.AvailabilityZone
east-2c

 &ReshardingConfiguration.ReshardingConfiguration.3.PreferredAvailabilityZones.AvailabilityZone
east-2d

 &ReshardingConfiguration.ReshardingConfiguration.4.PreferredAvailabilityZones.AvailabilityZone
east-2d

 &ReshardingConfiguration.ReshardingConfiguration.4.PreferredAvailabilityZones.AvailabilityZone
east-2c
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20171002T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see ModifyReplicationGroupShardConfiguration in the ElastiCache API Reference.

### Removing shards with online resharding

You can remove shards from your Valkey or Redis OSS (cluster mode enabled) cluster using the AWS Management Console, AWS CLI, or ElastiCache API.

### Topics

- Removing shards (Console)
- Removing shards (AWS CLI)
- Removing shards (ElastiCache API)

### Removing shards (Console)

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by removing shards using the AWS Management Console.

Before removing node groups (shards) from your replication group, ElastiCache makes sure that all your data will fit in the remaining shards. If the data will fit, the specified shards are deleted from the replication group as requested. If the data won't fit in the remaining node groups, the process is terminated and the replication group is left with the same node group configuration as before the request was made.

You can use the AWS Management Console to remove one or more shards from your Valkey or Redis OSS (cluster mode enabled) cluster. You cannot remove all the shards in a replication group. Instead, you must delete the replication group. For more information, see Deleting a replication group. The following procedure describes the process for deleting one or more shards.

**To remove shards from your Valkey or Redis OSS (cluster mode enabled) cluster**

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. Locate and choose the name, not the box to the left of the cluster's name, of the Valkey or Redis OSS (cluster mode enabled) cluster you want to remove shards from.

   > **Tip**
   >
   > Valkey or Redis OSS (cluster mode enabled) clusters have a value of 1 or greater in the **Shards** column.

4. From the list of shards, choose the box to the left of the name of each shard that you want to delete.
5. Choose **Delete shard**.

### Removing shards (AWS CLI)

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by removing shards using the AWS CLI.

> **Important**
>
> Before removing node groups (shards) from your replication group, ElastiCache makes sure that all your data will fit in the remaining shards. If the data will fit, the specified shards (`--node-groups-to-remove`) are deleted from the replication group as requested and their keyspaces mapped into the remaining shards. If the data will not fit in the remaining node groups, the process is terminated and the replication group is left with the same node group configuration as before the request was made.

You can use the AWS CLI to remove one or more shards from your Valkey or Redis OSS (cluster mode enabled) cluster. You cannot remove all the shards in a replication group. Instead, you must delete the replication group. For more information, see Deleting a replication group.

Use the following parameters with `modify-replication-group-shard-configuration`.

**Parameters**

- `--apply-immediately` – Required. Specifies the shard reconfiguration operation is to be started immediately.
- `--replication-group-id` – Required. Specifies which replication group (cluster) the shard reconfiguration operation is to be performed on.
- `--node-group-count` – Required. Specifies the number of shards (node groups) to exist when the operation is completed. When removing shards, the value of `--node-group-count` must be less than the current number of shards.

- `--node-groups-to-remove` – Required when `--node-group-count` is less than the current number of node groups (shards). A list of shard (node group) IDs to remove from the replication group.

The following procedure describes the process for deleting one or more shards.

### Example- Removing Shards

The following example removes two node groups from the Valkey or Redis OSS (cluster mode enabled) cluster `my-cluster`, so there are a total of two node groups when the operation completes. The keyspaces from the removed shards are distributed evenly over the remaining shards.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group-shard-configuration \
    --replication-group-id my-cluster \
    --node-group-count 2 \
    --node-groups-to-remove "0002" "0003" \
    --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group-shard-configuration ^
    --replication-group-id my-cluster ^
    --node-group-count 2 ^
    --node-groups-to-remove "0002" "0003" ^
    --apply-immediately
```

### Removing shards (ElastiCache API)

You can use the ElastiCache API to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster online by using the `ModifyReplicationGroupShardConfiguration` operation.

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by removing shards using the ElastiCache API.

> **Important**
>
> Before removing node groups (shards) from your replication group, ElastiCache makes sure that all your data will fit in the remaining shards. If the data will fit, the specified shards (`NodeGroupsToRemove`) are deleted from the replication group as requested and their keyspaces mapped into the remaining shards. If the data will not fit in the remaining node groups, the process is terminated and the replication group is left with the same node group configuration as before the request was made.

You can use the ElastiCache API to remove one or more shards from your Valkey or Redis OSS (cluster mode enabled) cluster. You cannot remove all the shards in a replication group. Instead, you must delete the replication group. For more information, see Deleting a replication group.

Use the following parameters with `ModifyReplicationGroupShardConfiguration`.

**Parameters**

- `ApplyImmediately=true` – Required. Specifies the shard reconfiguration operation is to be started immediately.
- `ReplicationGroupId` – Required. Specifies which replication group (cluster) the shard reconfiguration operation is to be performed on.
- `NodeGroupCount` – Required. Specifies the number of shards (node groups) to exist when the operation is completed. When removing shards, the value of `NodeGroupCount` must be less than the current number of shards.
- `NodeGroupsToRemove` – Required when `--node-group-count` is less than the current number of node groups (shards). A list of shard (node group) IDs to remove from the replication group.

The following procedure describes the process for deleting one or more shards.

**Example- Removing Shards**

The following example removes two node groups from the Valkey or Redis OSS (cluster mode enabled) cluster `my-cluster`, so there are a total of two node groups when the operation completes. The keyspaces from the removed shards are distributed evenly over the remaining shards.

```
https://elasticache.us-east-2.amazonaws.com/
    ?Action=ModifyReplicationGroupShardConfiguration
    &ApplyImmediately=true
    &NodeGroupCount=2
    &ReplicationGroupId=my-cluster
    &NodeGroupsToRemove.member.1=0002
    &NodeGroupsToRemove.member.2=0003
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20171002T192317Z
```

```
    &X-Amz-Credential=<credential>
```

### Online shard rebalancing

You can rebalance shards in your Valkey or Redis OSS (cluster mode enabled) cluster using the AWS Management Console, AWS CLI, or ElastiCache API.

**Topics**

- Online Shard Rebalancing (Console)
- Online shard rebalancing (AWS CLI)
- Online shard rebalancing (ElastiCache API)

### Online Shard Rebalancing (Console)

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by rebalancing shards using the AWS Management Console.

**To rebalance the keyspaces among the shards on your Valkey or Redis OSS (cluster mode enabled) cluster**

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. Choose the name, not the box to the left of the name, of the Valkey or Redis OSS (cluster mode enabled) cluster that you want to rebalance.

   > **Tip**
   >
   > Valkey or Redis OSS (cluster mode enabled) clusters have a value of 1 or greater in the **Shards** column.

4. Choose **Rebalance**.
5. When prompted, choose **Rebalance**. You might see a message similar to this one: *Slots in the replication group are uniformly distributed. Nothing to do. (Service: AmazonElastiCache; Status Code: 400; Error Code: InvalidReplicationGroupState; Request ID: 2246cebd-9721-11e7-8d5b-e1b0f086c8cf)*. If you do, choose **Cancel**.

### Online shard rebalancing (AWS CLI)

Use the following parameters with `modify-replication-group-shard-configuration`.

#### Parameters

- `-apply-immediately` – Required. Specifies the shard reconfiguration operation is to be started immediately.
- `--replication-group-id` – Required. Specifies which replication group (cluster) the shard reconfiguration operation is to be performed on.
- `--node-group-count` – Required. To rebalance the keyspaces across all shards in the cluster, this value must be the same as the current number of shards.

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by rebalancing shards using the AWS CLI.

#### Example- Rebalancing the Shards in a Cluster

The following example rebalances the slots in the Valkey or Redis OSS (cluster mode enabled) cluster `my-cluster` so that the slots are distributed as equally as possible. The value of `--node-group-count` (4) is the number of shards currently in the cluster.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group-shard-configuration \
    --replication-group-id my-cluster \
    --node-group-count 4 \
    --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group-shard-configuration ^
    --replication-group-id my-cluster ^
    --node-group-count 4 ^
    --apply-immediately
```

### Online shard rebalancing (ElastiCache API)

You can use the ElastiCache API to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster online by using the `ModifyReplicationGroupShardConfiguration` operation.

Use the following parameters with `ModifyReplicationGroupShardConfiguration`.

#### Parameters

- `ApplyImmediately=true` – Required. Specifies the shard reconfiguration operation is to be started immediately.
- `ReplicationGroupId` – Required. Specifies which replication group (cluster) the shard reconfiguration operation is to be performed on.
- `NodeGroupCount` – Required. To rebalance the keyspaces across all shards in the cluster, this value must be the same as the current number of shards.

The following process describes how to reconfigure the shards in your Valkey or Redis OSS (cluster mode enabled) cluster by rebalancing the shards using the ElastiCache API.

#### Example- Rebalancing a Cluster

The following example rebalances the slots in the Valkey or Redis OSS (cluster mode enabled) cluster `my-cluster` so that the slots are distributed as equally as possible. The value of `NodeGroupCount` (4) is the number of shards currently in the cluster.

```
https://elasticache.us-east-2.amazonaws.com/
    ?Action=ModifyReplicationGroupShardConfiguration
    &ApplyImmediately=true
    &NodeGroupCount=4
    &ReplicationGroupId=my-cluster
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20171002T192317Z
    &X-Amz-Credential=<credential>
```

### Online vertical scaling by modifying node type

By using online vertical scaling with Valkey and Redis OSS, you can scale your Valkey or Redis OSS clusters dynamically with minimal downtime. This allows your Valkey or Redis OSS cluster to serve requests even while scaling.

> **Note**
>
> Scaling is not supported between a data tiering cluster (for example, a cluster using an r6gd node type) and a cluster that does not use data tiering (for example, a cluster using an r6g node type). For more information, see Data tiering in ElastiCache.

You can do the following:

- **Scale up** – Increase read and write capacity by adjusting the node type of your Valkey or Redis OSS cluster to use a larger node type.

  ElastiCache dynamically resizes your cluster while remaining online and serving requests.
- **Scale down** – Reduce read and write capacity by adjusting the node type down to use a smaller node. Again, ElastiCache dynamically resizes your cluster while remaining online and serving requests. In this case, you reduce costs by downsizing the node.

> **Note**
>
> The scale up and scale down processes rely on creating clusters with newly selected node types and synchronizing the new nodes with the previous ones. To ensure a smooth scale up/down flow, do the following:
>
> - Ensure you have sufficient ENI (Elastic Network Interface) capacity. If scaling down, ensure the smaller node has sufficient memory to absorb expected traffic.
>
>   For best practices on memory management, see Managing reserved memory for Valkey and Redis OSS.
> - While the vertical scaling process is designed to remain fully online, it does rely on synchronizing data between the old node and the new node. We recommend that you initiate scale up/down during hours when you expect data traffic to be at its minimum.

- Test your application behavior during scaling in a staging environment, if possible.

**Contents**


- Online scaling up
  - Scaling up Valkey or Redis OSS clusters (Console)
  - Scaling up Valkey or Redis OSS clusters (AWS CLI)
  - Scaling up Valkey or Redis OSS clusters (ElastiCache API)
- Online scaling down
  - Scaling down Valkey or Redis OSS clusters (Console)
  - Scaling down Valkey or Redis OSS clusters (AWS CLI)
  - Scaling down Valkey or Redis OSS clusters (ElastiCache API)


## Online scaling up

**Topics**


- Scaling up Valkey or Redis OSS clusters (Console)
- Scaling up Valkey or Redis OSS clusters (AWS CLI)
- Scaling up Valkey or Redis OSS clusters (ElastiCache API)


### Scaling up Valkey or Redis OSS clusters (Console)

The following procedure describes how to scale up a Valkey or Redis OSS cluster using the ElastiCache Management Console. During this process, your cluster will continue to serve requests with minimal downtime.

**To scale up a Valkey or Redis OSS cluster (console)**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. From the list of clusters, choose the cluster.
4. Choose **Modify**.
5. In the **Modify Cluster** wizard:

- Choose the node type you want to scale to from the **Node type** list. To scale up, select a node type larger than your existing node.

6. If you want to perform the scale-up process right away, choose the **Apply immediately** box. If the **Apply immediately** box is not chosen, the scale-up process is performed during this cluster's next maintenance window.
7. Choose **Modify**.

   If you chose **Apply immediately** in the previous step, the cluster's status changes to *modifying*. When the status changes to *available*, the modification is complete and you can begin using the new cluster.

### Scaling up Valkey or Redis OSS clusters (AWS CLI)

The following procedure describes how to scale up a Valkey or Redis OSS cluster using the AWS CLI. During this process, your cluster will continue to serve requests with minimal downtime.

**To scale up a Valkey or Redis OSS cluster (AWS CLI)**

1. Determine the node types you can scale up to by running the AWS CLI `list-allowed-node-type-modifications` command with the following parameter.

   For Linux, macOS, or Unix:

   ```
   aws elasticache list-allowed-node-type-modifications \
        --replication-group-id my-replication-group-id
   ```

   For Windows:

   ```
   aws elasticache list-allowed-node-type-modifications ^
        --replication-group-id my-replication-group-id
   ```

   Output from the above command looks something like this (JSON format).

   ```
   {
        "ScaleUpModifications": [
            "cache.m5.2xlarge",
            "cache.m5.large",
            "cache.m5.xlarge",
   ```

```
          "cache.m4.10xlarge",
          "cache.m4.2xlarge",
          "cache.m4.4xlarge",
          "cache.m4.large",
          "cache.m4.xlarge",
          "cache.r5.2xlarge",
          "cache.r5.4xlarge",
          "cache.r5.12xlarge",
          "cache.r5.large",
          "cache.r5.xlarge"
     ]
        "ScaleDownModifications": [
          "cache.t2.micro",
          "cache.t2.small ",
          "cache.t2.medium"
     ],
}
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Modify your replication group to scale up to the new, larger node type, using the AWS CLI `modify-replication-group` command and the following parameters.

   - `--replication-group-id` – The name of the replication group you are scaling up to.
   - `--cache-node-type` – The new node type you want to scale the cluster. This value must be one of the node types returned by the `list-allowed-node-type-modifications` command in step 1.
   - `--cache-parameter-group-name` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
   - `--apply-immediately` – Causes the scale-up process to be applied immediately. To postpone the scale-up process to the cluster's next maintenance window, use the `--no-apply-immediately` parameter.

   For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group  \
     --replication-group-id my-redis-cluster \
     --cache-node-type cache.m5.xlarge \
```

```
    --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id my-redis-cluster ^
    --cache-node-type cache.m5.xlarge ^
    --apply-immediately
```

Output from the above command looks something like this (JSON format).

```
{
  "ReplicationGroup": {
        "Status": "modifying",
        "Description": "my-redis-cluster",
        "NodeGroups": [
            {
                "Status": "modifying",
                "Slots": "0-16383",
                "NodeGroupId": "0001",
                "NodeGroupMembers": [
                    {
                        "PreferredAvailabilityZone": "us-east-1f",
                        "CacheNodeId": "0001",
                        "CacheClusterId": "my-redis-cluster-0001-001"
                    },
                    {
                        "PreferredAvailabilityZone": "us-east-1d",
                        "CacheNodeId": "0001",
                        "CacheClusterId": "my-redis-cluster-0001-002"
                    }
                ]
            }
        ],
        "ConfigurationEndpoint": {
            "Port": 6379,
            "Address": "my-redis-
cluster.r7gdfi.clustercfg.use1.cache.amazonaws.com"
        },
        "ClusterEnabled": true,
        "ReplicationGroupId": "my-redis-cluster",
```

```
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "enabled",
        "SnapshotWindow": "07:30-08:30",
        "MemberClusters": [
            "my-redis-cluster-0001-001",
            "my-redis-cluster-0001-002"
        ],
        "CacheNodeType": "cache.m5.xlarge",
         "DataTiering": "disabled"
        "PendingModifiedValues": {}
    }
}
```

For more information, see modify-replication-group in the *AWS CLI Reference*.

3. If you used the `--apply-immediately`, check the status of the cluster using the AWS CLI `describe-cache-clusters` command with the following parameter. When the status changes to *available*, you can begin using the new, larger cluster node.

**Scaling up Valkey or Redis OSS clusters (ElastiCache API)**

The following process scales your cluster from its current node type to a new, larger node type using the ElastiCache API. During this process, ElastiCache updates the DNS entries so they point to the new nodes. Because of this you don't have to update the endpoints in your application. For Valkey 7.2 and above Redis OSS 5.0.5 and above, you can scale auto failover enabled clusters while the cluster continues to stay online and serve incoming requests. On version Redis OSS 4.0.10 and below, you may notice a brief interruption of reads and writes on previous versions from the primary node while the DNS entry is updated..

The amount of time it takes to scale up to a larger node type varies, depending upon your node type and the amount of data in your current cluster.

**To scale up a Valkey or Redis OSS Cache Cluster (ElastiCache API)**

1. Determine which node types you can scale up to using the ElastiCache API `ListAllowedNodeTypeModifications` action with the following parameter.

   - `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a specific replication group rather than all replication groups.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ListAllowedNodeTypeModifications
    &ReplicationGroupId=MyReplGroup
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see ListAllowedNodeTypeModifications in the *Amazon ElastiCache API Reference*.

2. Scale your current replication group up to the new node type using the `ModifyReplicationGroup` ElastiCache API action and with the following parameters.

   - `ReplicationGroupId` – the name of the replication group.
   - `CacheNodeType` – the new, larger node type of the clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in the previous step.
   - `CacheParameterGroupName` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
   - `ApplyImmediately` – Set to `true` to causes the scale-up process to be applied immediately. To postpone the scale-up process to the next maintenance window, use `ApplyImmediately=false`.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyReplicationGroup
    &ApplyImmediately=true
    &CacheNodeType=cache.m5.2xlarge
    &CacheParameterGroupName=redis7-cluster
    &ReplicationGroupId=myReplGroup
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &Version=2014-12-01
```

```
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ModifyReplicationGroup in the *Amazon ElastiCache API Reference*.

3. If you used `ApplyImmediately=true`, monitor the status of the replication group using the ElastiCache API `DescribeReplicationGroups` action with the following parameters. When the status changes from *modifying* to *available*, you can begin writing to your new, scaled up replication group.

   - `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a particular replication group rather than all replication groups.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeReplicationGroups
    &ReplicationGroupId=MyReplGroup
    &Version=2015-02-02
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

For more information, see DescribeReplicationGroups in the *Amazon ElastiCache API Reference*.

### Online scaling down

#### Topics

- Scaling down Valkey or Redis OSS clusters (Console)
- Scaling down Valkey or Redis OSS clusters (AWS CLI)
- Scaling down Valkey or Redis OSS clusters (ElastiCache API)

### Scaling down Valkey or Redis OSS clusters (Console)

The following procedure describes how to scale down a Valkey or Redis OSS cluster using the ElastiCache Management Console. During this process, your Valkey or Redis OSS cluster will continue to serve requests with minimal downtime.

#### To scale Down a Valkey or Redis OSS cluster (console)

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. From the list of clusters, choose your preferred cluster.
4. Choose **Modify**.
5. In the **Modify Cluster** wizard:
   - Choose the node type you want to scale to from the **Node type** list. To scale down, select a node type smaller than your existing node. Note that not all node types are available to scale down to.
6. If you want to perform the scale down process right away, choose the **Apply immediately** box. If the **Apply immediately** box is not chosen, the scale-down process is performed during this cluster's next maintenance window.
7. Choose **Modify**.

   If you chose **Apply immediately** in the previous step, the cluster's status changes to *modifying*. When the status changes to *available*, the modification is complete and you can begin using the new cluster.

### Scaling down Valkey or Redis OSS clusters (AWS CLI)

The following procedure describes how to scale down a Valkey or Redis OSS cluster using the AWS CLI. During this process, your cluster will continue to serve requests with minimal downtime.

#### To scale down a Valkey or Redis OSS cluster (AWS CLI)

1. Determine the node types you can scale down to by running the AWS CLI `list-allowed-node-type-modifications` command with the following parameter.

   For Linux, macOS, or Unix:

```
aws elasticache list-allowed-node-type-modifications \
     --replication-group-id my-replication-group-id
```

For Windows:

```
aws elasticache list-allowed-node-type-modifications ^
     --replication-group-id my-replication-group-id
```

Output from the above command looks something like this (JSON format).

```
{
     "ScaleUpModifications": [
         "cache.m5.2xlarge",
         "cache.m5.large",
         "cache.m5.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r5.2xlarge",
         "cache.r5.4xlarge",
         "cache.r5.12xlarge",
         "cache.r5.large",
         "cache.r5.xlarge"
     ]

        "ScaleDownModifications": [
         "cache.t2.micro",
         "cache.t2.small ",
         "cache.t2.medium "
     ]
}
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Modify your replication group to scale down to the new, smaller node type, using the AWS CLI `modify-replication-group` command and the following parameters.

   - `--replication-group-id` – The name of the replication group you are scaling down to.

- `--cache-node-type` – The new node type you want to scale the cluster. This value must be one of the node types returned by the `list-allowed-node-type-modifications` command in step 1.
- `--cache-parameter-group-name` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.
- `--apply-immediately` – Causes the scale-up process to be applied immediately. To postpone the scale-down process to the cluster's next maintenance window, use the `--no-apply-immediately` parameter.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group  \
     --replication-group-id my-redis-cluster \
     --cache-node-type cache.t2.micro \
     --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
     --replication-group-id my-redis-cluster ^
     --cache-node-type cache.t2.micro ^
     --apply-immediately
```

Output from the above command looks something like this (JSON format).

```
{
  "ReplicationGroup": {
        "Status": "modifying",
        "Description": "my-redis-cluster",
        "NodeGroups": [
            {
                "Status": "modifying",
                "Slots": "0-16383",
                "NodeGroupId": "0001",
                "NodeGroupMembers": [
```

```
                    {
                        "PreferredAvailabilityZone": "us-east-1f",
                        "CacheNodeId": "0001",
                        "CacheClusterId": "my-redis-cluster-0001-001"
                    },
                    {
                        "PreferredAvailabilityZone": "us-east-1d",
                        "CacheNodeId": "0001",
                        "CacheClusterId": "my-redis-cluster-0001-002"
                    }
                ]
            }
        ],
        "ConfigurationEndpoint": {
            "Port": 6379,
            "Address": "my-redis-
cluster.r7gdfi.clustercfg.use1.cache.amazonaws.com"
        },
        "ClusterEnabled": true,
        "ReplicationGroupId": "my-redis-cluster",
        "SnapshotRetentionLimit": 1,
        "AutomaticFailover": "enabled",
        "SnapshotWindow": "07:30-08:30",
        "MemberClusters": [
            "my-redis-cluster-0001-001",
            "my-redis-cluster-0001-002"
        ],
        "CacheNodeType": "cache.t2.micro",
         "DataTiering": "disabled"
        "PendingModifiedValues": {}
    }
}
```

For more information, see modify-replication-group in the *AWS CLI Reference*.

3. If you used the `--apply-immediately`, check the status of the cluster using the AWS CLI `describe-cache-clusters` command with the following parameter. When the status changes to *available*, you can begin using the new, smaller cluster node.

### Scaling down Valkey or Redis OSS clusters (ElastiCache API)

The following process scales your replication group from its current node type to a new, smaller node type using the ElastiCache API. During this process, your Valkey or Redis OSS cluster will continue to serve requests with minimal downtime.

The amount of time it takes to scale down to a smaller node type varies, depending upon your node type and the amount of data in your current cluster.

#### Scaling down (ElastiCache API)

1. Determine which node types you can scale down to using the ElastiCache API `ListAllowedNodeTypeModifications` action with the following parameter.

   - `ReplicationGroupId` – the name of the replication group. Use this parameter to describe a specific replication group rather than all replication groups.

   ```
   https://elasticache.us-west-2.amazonaws.com/
       ?Action=ListAllowedNodeTypeModifications
       &ReplicationGroupId=MyReplGroup
       &Version=2015-02-02
       &SignatureVersion=4
       &SignatureMethod=HmacSHA256
       &Timestamp=20150202T192317Z
       &X-Amz-Credential=<credential>
   ```

   For more information, see ListAllowedNodeTypeModifications in the *Amazon ElastiCache API Reference*.

2. Scale your current replication group down to the new node type using the `ModifyReplicationGroup` ElastiCache API action and with the following parameters.

   - `ReplicationGroupId` – the name of the replication group.
   - `CacheNodeType` – the new, smaller node type of the clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in the previous step.
   - `CacheParameterGroupName` – [Optional] Use this parameter if you are using `reserved-memory` to manage your cluster's reserved memory. Specify a custom cache parameter

group that reserves the correct amount of memory for your new node type. If you are using `reserved-memory-percent` you can omit this parameter.

- `ApplyImmediately` – Set to `true` to causes the scale-down process to be applied immediately. To postpone the scale-down process to the next maintenance window, use `ApplyImmediately=false`.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyReplicationGroup
    &ApplyImmediately=true
    &CacheNodeType=cache.t2.micro
    &CacheParameterGroupName=redis7-cluster
    &ReplicationGroupId=myReplGroup
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &Version=2014-12-01
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ModifyReplicationGroup in the *Amazon ElastiCache API Reference*.

# Getting started with Bloom filters

ElastiCache supports the Bloom filter data structure, which provides a space efficient probabilistic data structure to check if an element is a member of a set. When using Bloom filters, false positives are possible—a filter can incorrectly indicate that an element exists, even though that element was not added to the set. However, using Bloom filters will prevent false *negatives*—incorrect indications that an element does *not* exist, even though that element was added to the set.

You can set the percentage of potential false positives to a preferred rate for your workload, by adjusting the fp rate. You can also configure capacity (the number of items a Bloom filter can hold), scaling and non-scaling properties, and more.

After you create a cluster with a supported engine version, the Bloom data type and associated commands are automatically available. The `bloom` data type is API compatible with the Bloom

filter command syntax of the official Valkey client libraries including `valkey-py`, `valkey-java`, and `valkey-go`. You can easily migrate existing Bloom-based Valkey and Redis OSS applications into ElastiCache. For a complete list of commands see Bloom filter commands.

The Bloom-related metrics `BloomFilterBasedCmds`, `BloomFilterBasedCmdsLatency`, and `BloomFilterBasedCmdsECPUs` are incorporated into CloudWatch to monitor the usage of this data type. For more information, see Metrics for Valkey and Redis OSS.

**Note**

- To use Bloom filters, you must be running on ElastiCache Valkey 8.1 and later.
- The bloom data type is not RDB compatible with other non-Valkey based bloom offerings.

## Bloom filters data type overview

Bloom filters are a space efficient probabilistic data structure that allows adding elements and checking whether elements exist. False positives are possible where a filter incorrectly indicates that an element exists, even though it was not added. However, Bloom Filters guarantee that false negatives (incorrectly indicating that an element does not exist, even though it was added) do not occur.

The main source of documentation for bloom filters can be found on the valkey.io documentation page. This contains the following information:

- Common use cases for bloom filters
  - Advertisement / Event deduplication
  - Fraud detection
  - Filtering harmful content / spam
  - Unique user detection
- Differences between scaling and non scaling bloom filters
  - How to decide between scaling and non scaling bloom filters
- Bloom properties
  - Learn about the tunable properties of Bloom filters. This includes the false positive rate, capacity, scaling and non scaling properties, and more.

- Performance of bloom commands
- Monitoring overall bloom filter stats
- Handling large bloom filters
  - Recommendations and details on how to check if a bloom filter is reaching its memory usage limit, and if it can scale to reach the desired capacity.
  - You can specifically check the amount of memory consumed by a bloom filter document through using the BF.INFO command.

## Bloom size limit

The consumption of memory by a single Bloom filter object is limited to 128 MB. You can check the amount of memory consumed by a Bloom filter by using the `BF.INFO <key> SIZE` command.

## Bloom ACLs

Similar to the existing per-datatype categories (@string, @hash, etc.) a new category @bloom is added to simplify managing access to Bloom commands and data. No other existing Valkey or Redis OSS commands are members of the @bloom category.

There are 3 existing ACL categories that are updated to include the new Bloom commands: @read, @write and @fast. The following table indicates the mapping of Bloom commands to the appropriate categories.

| Bloom command | @read | @write | @fast | @bloom |
|---|---|---|---|---|
| BF.ADD | | y | y | y |
| BF.CARD | y | | y | y |
| BF.EXISTS | y | | y | y |
| BF.INFO | y | | y | y |
| BF.INSERT | | y | y | y |
| BF.MADD | | y | y | y |

| Bloom command | @read | @write | @fast | @bloom |
|---|---|---|---|---|
| BF.MEXISTS | y | | y | y |
| BF.RESERVE | y | | y | y |

## Bloom filter related metrics

The following CloudWatch metrics related to bloom data structures are provided:

| CW Metrics | Unit | Serverless/Node-based | Description |
|---|---|---|---|
| BloomFilterBasedCmds | Count | Both | The total number of Bloom filter commands, including both read and write commands. |
| BloomFilterBasedCmdsLatency | Microseconds | Self-managed | Latency of all Bloom filter commands, including both read and write commands. |
| BloomFilterBasedCmdsECPUs | Count | Serverless | ECPUs consumed by all Bloom filter commands, including both read and write commands. |

## Bloom filter commands

Bloom Filter commands are documented on the Valkey.io website. Each command page provides a comprehensive overview of the bloom commands, including its syntax, behavior, return values, and potential error conditions.

| Name | Description |
| --- | --- |
| BF.ADD | Adds a single item to a bloom filter.If the filter doesn't already exist, it is created. |
| BF.CARD | Returns the cardinality of a bloom filter. |
| BF.EXISTS | Determines if the bloom filter contains the specified item. |
| BF.INFO | Returns usage information and properties of a specific bloom filter. |
| BF.INSERT | Creates a bloom filter with 0 or more itemes, or adds items to an existing bloom filter. |
| BF.MADD | Adds one or more items to a bloom filter. |
| BF.MEXISTS | Determines if the bloom filter contains 1 or more items. |
| BF.RESERVE | Creates an empty bloom filter with the specified properties. |

**Note**

**BF.LOAD** is not supported by ElastiCache. It is only relevant for AOF usage, which ElastiCache does not support.

## Getting started with Watch in Serverless

ElastiCache supports the `WATCH` command, which allows you to monitor keys for changes and execute conditional transactions. The `WATCH` command is particularly useful for applications that require optimistic concurrency control, ensuring that transactions are only executed if the monitored keys have not been modified. This includes modifications made by a client, like write commands, and by Valkey itself, like expiration or eviction. If keys were modified from the time they were set in `WATCH` and by the time `EXEC` is received, the entire transaction will be aborted.

For ElastiCache Serverless, the following constraints are introduced -

ElastiCache Serverless `WATCH` is scoped to a single hash slot. That means only keys that map to the same hash slot can be watched at the same time by the same connection, and the transaction that follows the watch commands can only operate on the same hash slot. When an application attempts to watch keys from different hash slots, or execute transaction commands that operate on keys mapped to a different hash slot than the watched keys', a `CROSSSLOT` error will be returned. Hash tags can be used to ensure multiple keys are mapped to the same hash slot.

Additionally, `SCAN` command cannot be executed inside a connection with watched keys and will return `command not supported during watch state` error.

The transaction will be aborted (as if watched keys were touched) when ElastiCache Serverless has no certainty of whether a key was modified. For example, when a slot has been migrated and the watched keys cannot be found on the same node.

**Code Examples**

## Watch and Operate on Keys from Different Slots

In the following example, the watched key and the key specified in the `SET` command map to different hash slots. The execution returns a `CROSSSLOT ERROR`.

```
> WATCH foo:{005119}
OK
> MULTI
OK
> SET bar:{011794} 1234
QUEUED
> EXEC
CROSSSLOT Keys in request don't hash to the same slot
```

## Watch and Operate on Keys from the Same Slot

The following example shows a successful transaction, as The key set in `WATCH` wasn't changed.

```
> WATCH foo:{005119}
OK
```

```
> MULTI
OK
> SET bar:{005119} 1234
QUEUED
> EXEC
1) OK
```

### Watch keys from Different Slots

In the following example, an attempt to WATCH keys from different slots simultaneously within the same client connection returns a CROSSSLOT ERROR.

```
> WATCH foo:{005119}
OK
> WATCH bar:{123455}
CROSSSLOT Keys in request don't hash to the same slot
```

## Watch limit

Every client connection can watch up to 1000 keys at the same time.

## Supported commands related to Watch

WATCH and UNWATCH commands are documented on the Valkey.io website. It provides a comprehensive overview of the commands, including its syntax, behavior, return values, and potential error conditions.

# Getting started with Search

Amazon ElastiCache for Valkey supports real-time full-text, exact-match (tag), numeric range, vector, and hybrid searches. It offers a high-performance search engine optimized for AI-driven / Search / Analytics / Recommendation System related workloads, delivering latency as low as microseconds. ElastiCache for Valkey provides the lowest latency vector search with the highest throughput and best price-performance at 95%+ recall rate among popular vector databases on AWS. ElastiCache for Valkey provides capabilities to index, search, and update billions of high-dimensional vector embeddings from popular providers like Amazon Bedrock, Amazon SageMaker, Anthropic or OpenAI for fast search and retrieval with up to 99% recall. Vector search for Amazon

ElastiCache is ideal for use cases where peak performance, consistency and scalability are the most important selection criteria. This includes semantic caching, AI agent memory, retrieval-augmented generation, real-time recommendations, personalization, and anomaly detection.

You can use search on node-based clusters with Valkey 8.2 or later and on ElastiCache Serverless caches with Valkey 9.0 or later. To learn more about the Search feature availability in ElastiCache Valkey engine versions, see Search features and limits.

For a detailed description of the supported commands, examples and configuration options, see the Command Reference.

You can learn more about the Search feature and related concepts at Valkey Search documentation.

> **Note**
>
> To build search workflows with AI agents, visit the Agent tools for ElastiCache page to install the ElastiCache Skill and Valkey MCP server. They provide the knowledge as well as tools to manage indexes, generate vector embeddings, and a unified search tool for setting up search workflows.

**Topics**


- Search features and limits
- Choosing the appropriate configuration
- Search write throttling


# Search features and limits

## Search availability

ElastiCache Valkey version 9.0 and above provides support for pure non vector, vector, and hybrid workloads including Numeric, Tag (exact-match), Full-text, Vector searches, and Aggregations on node-based clusters in all AWS Regions at no additional cost, and on ElastiCache Serverless caches in all AWS Regions, billed through ElastiCache Processing Units (ECPUs) and data stored.

ElastiCache Valkey version 8.2 provides support for Vector Search on node-based clusters in all AWS Regions at no additional cost.

You can also use search on your existing clusters by upgrading from any version of Valkey, or Redis OSS to Valkey versions mentioned above, in a few clicks with no downtime.

Search is currently available on all ElastiCache instance types other than nodes with data tiering. Using search on t3 and t4g instances requires increasing the memory reserve to at least 50% for micro and 30% for small instances. See this page to find out more.

## Valkey Search versions

Search on ElastiCache for Valkey 9.0 relies on Valkey Search 1.2 and does not support the FT.HYBRID command. Search on ElastiCache for Valkey 8.2 relies on Valkey Search 1.0 and does not support text, tag, or numeric queries independent of vectors.

## Parametric restrictions

The following table shows limits for various search items:

**Search limits**

| Item | Maximum value (Serverless) | Maximum value (node-based 9.0+) | Maximum value (node-based 8.2) |
|---|---|---|---|
| Number of dimensions in a vector | 4096 | 32768 | 32768 |
| Number of indexes that can be created | 1000 | 1000 | 10 |
| Number of fields in an index | 1000 | 1000 | 50 |
| FT.SEARCH TIMEOUT clause (milliseconds) | 60000 | 60000 | 60000 |
| Maximum number of prefixes allowed per index | 8 | 16 | 16 |
| Maximum length of a tag field | 256 | 10000 | 10000 |

| Item | Maximum value (Serverless) | Maximum value (node-based 9.0+) | Maximum value (node-based 8.2) |
|---|---|---|---|
| Maximum length of a numeric field | 128 | 256 | 256 |
| HNSW M parameter | 32 | 2000000 | 2000000 |
| HNSW EF_CONSTRUCTION parameter | 256 | 1000000 | 4096 |
| HNSW EF_RUNTIME parameter | 512 | 1000000 | 4096 |
| Number of terms allowed to be used in a query string in FT.SEARCH/ FT.AGGREGATE commands | 1000 | 1000 | 16 |
| Number of text attributes allowed per index | 64 | 64 | NA |
| Maximum text word expansions in Prefix, Suffix, Fuzzy, and Stem Term Searches | 200 | 200 | NA |
| K value in a vector (KNN) query | 512 | 10000 | 128 |

## Operational restrictions

### Index Persistence and Backfilling

You can read more about this at Valkey search index creation and backfill.

### Scaling limits

In ElastiCache Valkey version 9.0, during scaling events, the write RPS may reduce for the duration of the event. In ElastiCache Valkey version 8.2, during scaling events, the index may undergo backfill as data is migrated and this will result in a reduced recall for search queries.

### Snapshot import/export and Live Migration

The RDB files from the one cluster with search indexes can be imported to another ElastiCache Valkey cluster with version 8.2 or higher, or to an ElastiCache Serverless cache with Valkey 9.0 or higher. The new cluster will rebuild the index content on loading the RDB file. However, the presence of search indexes in an RDB file limits the compatibility of that data with prior versions of Valkey. The format of the search indexes defined by the vector search functionality is only understood by another ElastiCache cluster with Valkey version 8.2 or higher. However, RDB files that do not contain indexes are not restricted in this fashion.

### Out of Memory during backfill

Similar to Valkey OSS write operations, an index backfill is subjected to out-of-memory limitations. If engine memory is filled up while a backfill is in progress, all backfills are paused. If memory becomes available, the backfill process is resumed. It is possible to delete an index when backfill is paused due to out of memory.

### Write throttling with durability enabled

When durability is enabled and search indexes are configured, write commands targeting indexed keys may be throttled to maintain transactional log performance. For details, see Search write throttling.

### Transactions

The commands `FT.CREATE` and `FT.DROPINDEX`, cannot be executed in a transactional context, i.e., not within a `MULTI/EXEC` block or within a LUA or FUNCTION script. Additionally, the `FT.SEARCH` and `FT.AGGREGATE` commands cannot be executed in a transactional context in a ElastiCache Valkey Cluster operating in Cluster Mode.

## Search security

Valkey ACL (Access Control Lists) security mechanisms for both command and data access are extended to control the search facility. ACL control of individual search commands is fully supported. A new ACL category, `@search`, is provided and many of the existing categories (`@fast`,

`@read`, `@write`, etc.) are updated to include the new commands. Search commands do not modify key data, meaning that the existing ACL machinery for write access is preserved. The access rules for `HASH` and `JSON` operations are not modified by the presence of an index; normal key-level access control is still applied to those commands.

Search commands with an index also have their access controlled through ACL. Access checks are performed at the whole-index level, not at the per-key level. This means that access to an index is granted to a user only if that user has permission to access all possible keys within the keyspace prefix list of that index. In other words, the actual contents of an index don't control the access. Rather, it is the theoretical contents of an index as defined by the prefix list which is used for the security check. Situations where a user has read and/or write access to a key but is unable to access an index containing that key are possible. Note that only read access to the keyspace is required to create or use an index – the presence or absence of write access is not considered.

# Choosing the appropriate configuration

## Choosing between Serverless and node-based

Choose Serverless when your traffic is hard to predict, your index holds tens of millions of vectors or documents, and you want to start quickly without planning capacity. Choose a node-based cluster when you need the lowest latency on hundreds of millions of vectors, sustained high query throughput, or fast bulk loads of large vector indexes. With ElastiCache Serverless, you don't choose an instance type, because Serverless scales memory and compute with your search workload.

For node-based clusters, choose the right instance type based on the memory and cpu requirements of your search workload.

## Memory consumption

Memory consumption for Vector fields is based on the number of vectors, the number of dimensions, the M-value, and the amount of non-vector data, such as metadata associated to the vector or other data stored within the instance. The total memory required is a combination of the space needed for the actual vector data, and the space required for the vector indices. The space required for Vector data is calculated by measuring the actual capacity required for storing vectors within `HASH` or `JSON` data structures and the overhead to the nearest memory slabs, for optimal memory allocations. Each of the vector indexes uses references to the vector data stored in these data structures as well as an additional copy of the vector in the index. It is advised to plan for this additional space consumption by the index.

The number of vectors depend on how you decide to represent your data as vectors. For instance, you can choose to represent a single document into several chunks, where each chunk represents a vector. Alternatively, you could choose to represent the whole document as a single vector. The number of dimensions of your vectors is dependent on the embedding model you choose. For instance, if you choose to use the AWS Titan embedding model then the number of dimensions would be 1536. Note that you should test the instance type to make sure it fits your requirements.

### Scaling your workload

Search supports all three methods of scaling: horizontal, vertical and replicas. When scaling for capacity, vector search behaves just like regular Valkey, i.e., increasing the memory of individual nodes (vertical scaling) or increasing the number of nodes (horizontal scaling) will increase the overall capacity. In cluster mode, the `FT.CREATE` command can be sent to any primary node of the cluster and the system will automatically distribute the new index definition to all cluster members.

However, from a performance perspective, search behaves very differently from regular Valkey. The multi-threaded implementation of search means that additional CPUs yield up to linear increases in both query and ingestion throughput. Horizontal scaling yields linear increases in ingestion throughput but may reduce query throughput. If additional query throughput is required, scaling through replicas or additional CPUs is required.

## Search write throttling

Search write throttling applies only to node-based clusters with durability enabled. On ElastiCache Serverless, search commands follow the same ElastiCache Processing Unit (ECPU) limits as all other commands. To maintain optimal performance and data durability, ElastiCache in **durable mode** implements write throttling on search traffic when necessary. Throttling helps ensure that automatic backup mechanisms operate effectively without falling behind during periods of high write activity. By temporarily reducing write throughput, the system preserves the integrity of the Multi-AZ transactional log, which is essential for fast database recovery and restart.

### Scope of throttling

Only write commands targeting keys that belong to a search index are throttled. Writes to non-indexed keys and all read commands are **not** affected.

The following commands are subject to throttling when they target indexed keys:

**Commands subject to search write throttling**

| Category | Commands |
|---|---|
| Hash | HSET, HSETNX, HMSET, HINCRBY, HINCRBYFLOAT , HDEL |
| JSON | JSON.SET, JSON.DEL, JSON.NUMINCRBY , JSON.NUMMULTBY , JSON.STRAPPEND , JSON.ARRAPPEND , JSON.ARRINSERT , JSON.ARRPOP , JSON.ARRTRIM , JSON.TOGGLE , JSON.CLEAR , JSON.MERGE |
| Generic | DEL, UNLINK, RENAME, RENAMENX, COPY, RESTORE |

## What clients experience

Throttled commands are **delayed, not rejected**. Affected writes take longer to complete but still succeed. No errors are returned to the client.

You can observe the impact through the following Amazon CloudWatch metrics:

- `SuccessfulWriteRequestLatency` and `SearchBasedSetCmdsLatency` — Reflect increased latency on affected writes.
- `SearchWriteThrottleActive`, `SearchWriteThrottledClientsCount`, and `SearchWriteThrottleEvents` — Indicate whether throttling is active and to what degree. See Monitoring for details.

## When throttling activates

The system monitors CPU usage of the search module's writer threads over a rolling 2-hour window. Throttling activates when the average CPU usage during that window exceeds 50%, and adjusts the allowed write rate to bring average utilization back to 50%.

Because the metric is averaged over a 2-hour window, short bursts of elevated CPU usage do not trigger throttling on their own — as long as they are offset by lower usage within the same window.

To prevent over-throttling, the system also evaluates current CPU usage in real time. If current CPU usage is already at 50% or below, the system holds the write rate steady rather than reducing it

further, even if the 2-hour average remains above the threshold. This ensures that write capacity never falls below 50% of normal throughput.

## When throttling deactivates

Once the 2-hour average CPU usage drops below 50%, the system gradually increases the allowed write rate until full throughput is restored and throttling deactivates.

## Monitoring

The following Amazon CloudWatch metrics are available for monitoring search write throttling:

**Search write throttling CloudWatch metrics**

| Metric | Description | Unit |
|---|---|---|
| `SearchWriteThrottleActive` | Indicates whether throttling is currently active. 1 = active, `0` = inactive. | Boolean |
| `SearchWriteThrottledClientsCount` | The number of client connections currently being throttled. | Count |
| `SearchWriteThrottleEvents` | The number of throttle events within the reporting interval. | Count |
| `SearchWriteCPUUtilization` | Current CPU utilization of the search writer threads. | Percent |

## Best practices

- **Monitor `SearchWriteCPUUtilization`** — Track your search write CPU usage to understand your workload patterns and anticipate when you might approach the throttling threshold.
- **Monitor `SearchWriteThrottleActive`** — Track whether throttling is active so you can investigate and respond promptly.
- **Plan sustained ingestion around the 2-hour window** — The system uses a 2-hour rolling average, so short bursts of high write activity are fully supported as long as they are offset by lower usage within the same window.

- **Scale your cluster if you observe sustained or frequent throttling** — If your workload consistently exceeds the threshold and throttling impacts your application's latency requirements, consider scaling to add capacity.

# Getting started with data aggregations

Amazon ElastiCache for Valkey supports aggregation queries, enabling you to filter, group, transform, and summarize data stored in your cache with a single query. Aggregations run server-side in memory, performing computations directly on indexed data and returning only the final results to the client. This eliminates the need to transfer large datasets to the application layer for processing, reducing network overhead and enabling response latencies as low as microseconds over terabytes of data.

Aggregation queries are built as pipelines of chained stages, where the output of each stage feeds into the next. Available stages include `GROUPBY` for grouping results by field values, `REDUCE` for applying functions such as COUNT, SUM, AVG, MIN, and MAX to each group, `APPLY` for computing derived values using expressions, `FILTER` for pruning intermediate results, `SORTBY` for ordering output, `LIMIT` for controlling result set size, and `LOAD` for pulling additional hash fields into the pipeline at query time. You can combine these stages in any order and repeat them to construct multi-step analytical workflows within a single command.

Common use cases include:

- **Faceted search** - Group matching catalog items by attributes such as category, brand, or rating and return counts per group to power dynamic filter UIs.
- **Real-time rankings and trends** - Compute top performers by engagement metrics such as views, scores, or sales across categories, with results that reflect the latest data without scheduled recomputation.
- **Operational reporting and analytics** - Generate summary metrics such as averages, totals, and distributions over session, order, or transaction data on demand or on a schedule, without a separate analytics layer.

For more details on the command syntax, see FT.AGGREGATE on the Valkey documentation.

For more details related to aggregation expressions, see Valkey search expressions.

# Getting started with JSON for Valkey and Redis OSS

ElastiCache supports the native JavaScript Object Notation (JSON) format, which is a simple, schemaless way to encode complex datasets inside Valkey and Redis OSS clusters. You can natively store and access data using the JavaScript Object Notation (JSON) format inside the clusters, and update JSON data stored in those clusters—without needing to manage custom code to serialize and deserialize it.

In addition to using Valkey and Redis OSS API operations for applications that operate over JSON, you can now efficiently retrieve and update specific portions of a JSON document without needing to manipulate the entire object. This can improve performance and reduce cost. You can also search your JSON document contents using the Goessner-style `JSONPath` query.

After you create a cluster with a supported engine version, the JSON data type and associated commands are automatically available. API compatible and RDB compatible with version 2 of the JSON module, so you can easily migrate existing JSON-based Valkey and Redis OSS applications into ElastiCache. For more information on the supported commands, see Supported Valkey and Redis OSS commands.

The JSON-related metrics `JsonBasedCmds` and `JsonBasedCmdsLatency` are incorporated into CloudWatch to monitor the usage of this data type. For more information, see Metrics for Valkey and Redis OSS.

> **Note**
>
> To use JSON, you must be running Valkey 7.2 and later, or Redis OSS 6.2.6 or later.

**Topics**

- JSON data type overview
- Supported Valkey and Redis OSS commands

## JSON data type overview

ElastiCache supports a number of Valkey and Redis OSS commands for working with the JSON data type. The following is an overview of the JSON data type and a detailed list of commands that are supported.

## Terminology

| Term | Description |
| --- | --- |
| JSON document | Refers to the value of a JSON key. |
| JSON value | Refers to a subset of a JSON document, including the root that represents the entire document. A value could be a container or an entry within a container. |
| JSON element | Equivalent to JSON value. |

## Supported JSON standard

JSON format is compliant with RFC 7159 and ECMA-404 JSON data interchange standard. UTF-8 Unicode in JSON text is supported.

## Root element

The root element can be of any JSON data type. Note that in earlier RFC 4627, only objects or arrays were allowed as root values. Since the update to RFC 7159, the root of a JSON document can be of any JSON data type.

## Document size limit

JSON documents are stored internally in a format that's optimized for rapid access and modification. This format typically results in consuming somewhat more memory than the equivalent serialized representation of the same document.

The consumption of memory by a single JSON document is limited to 64 MB, which is the size of the in-memory data structure, not the JSON string. You can check the amount of memory consumed by a JSON document by using the `JSON.DEBUG MEMORY` command.

## JSON ACLs

- Similar to the existing per-datatype categories (@string, @hash, etc.), a new category @json is added to simplify managing access to JSON commands and data. No other existing Valkey

or Redis OSS commands are members of the @json category. All JSON commands enforce any keyspace or command restrictions and permissions.

- There are five existing Valkey and Redis OSS ACL categories that are updated to include the new JSON commands: @read, @write, @fast, @slow and @admin. The following table indicates the mapping of JSON commands to the appropriate categories.

**ACL**

| JSON command | @read | @write | @fast | @slow | @admin |
|---|---|---|---|---|---|
| JSON.ARRAPPEND | | y | y | | |
| JSON.ARRINDEX | y | | y | | |
| JSON.ARRINSERT | | y | y | | |
| JSON.ARRLEN | y | | y | | |
| JSON.ARRPOP | | y | y | | |
| JSON.ARRTRIM | | y | y | | |
| JSON.CLEAR | | y | y | | |
| JSON.DEBUG | y | | | y | y |
| JSON.DEL | | y | y | | |
| JSON.FORGET | | y | y | | |
| JSON.GET | y | | y | | |

| JSON command | @read | @write | @fast | @slow | @admin |
|---|---|---|---|---|---|
| JSON.MGET | y | | y | | |
| JSON.NUMINCRBY | | y | y | | |
| JSON.NUMMULTBY | | y | y | | |
| JSON.OBJKEYS | y | | y | | |
| JSON.OBJLEN | y | | y | | |
| JSON.RESP | y | | y | | |
| JSON.SET | | y | | y | |
| JSON.STRAPPEND | | y | y | | |
| JSON.STRLEN | y | | y | | |
| JSON.STRLEN | y | | y | | |
| JSON.TOGGLE | | y | y | | |
| JSON.TYPE | y | | y | | |
| JSON.NUMINCRBY | | y | y | | |

## Nesting depth limit

When a JSON object or array has an element that is itself another JSON object or array, that inner object or array is said to “nest” within the outer object or array. The maximum nesting depth limit is 128. Any attempt to create a document that contains a nesting depth greater than 128 will be rejected with an error.

## Command syntax

Most commands require a key name as the first argument. Some commands also have a path argument. The path argument defaults to the root if it's optional and not provided.

Notation:

- Required arguments are enclosed in angle brackets. For example: <key>
- Optional arguments are enclosed in square brackets. For example: [path]
- Additional optional arguments are indicated by an ellipsis ("…"). For example: [json ...]

## Path syntax

Redis JSON supports two kinds of path syntaxes:

- **Enhanced syntax** – Follows the JSONPath syntax described by Goessner, as shown in the following table. We've reordered and modified the descriptions in the table for clarity.
- **Restricted syntax** – Has limited query capabilities.

> **Note**
>
> Results of some commands are sensitive to which type of path syntax is used.

If a query path starts with '$', it uses the enhanced syntax. Otherwise, the restricted syntax is used.

### Enhanced syntax

| Symbol/Expression | Description |
| --- | --- |
| $ | The root element. |

| Symbol/Expression | Description |
| --- | --- |
| . or [] | Child operator. |
| .. | Recursive descent. |
| * | Wildcard. All elements in an object or array. |
| [] | Array subscript operator. Index is 0-based. |
| [,] | Union operator. |
| [start:end:step] | Array slice operator. |
| ?() | Applies a filter (script) expression to the current array or object. |
| () | Filter expression. |
| @ | Used in filter expressions that refer to the current node being processed. |
| == | Equal to, used in filter expressions. |
| != | Not equal to, used in filter expressions. |
| > | Greater than, used in filter expressions. |
| >= | Greater than or equal to, used in filter expressions. |
| < | Less than, used in filter expressions. |
| <= | Less than or equal to, used in filter expressio ns. |
| && | Logical AND, used to combine multiple filter expressions. |
| \|\| | Logical OR, used to combine multiple filter expressions. |

### Examples

The following examples are built on Goessner's example XML data, which we have modified by adding additional fields.

```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95,
        "in-stock": true,
        "sold": true
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99,
        "in-stock": false,
        "sold": true
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99,
        "in-stock": true,
        "sold": false
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99,
        "in-stock": false,
        "sold": false
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95,
      "in-stock": true,
      "sold": false
    }
```

```
  }
}
```

| Path | Description |
| --- | --- |
| $.store.book[*].author | The authors of all books in the store. |
| $..author | All authors. |
| $.store.* | All members of the store. |
| $["store"].* | All members of the store. |
| $.store..price | The price of everything in the store. |
| $..* | All recursive members of the JSON structure. |
| $..book[*] | All books. |
| $..book[0] | The first book. |
| $..book[-1] | The last book. |
| $..book[0:2] | The first two books. |
| $..book[0,1] | The first two books. |
| $..book[0:4] | Books from index 0 to 3 (ending index is not inclusive). |
| $..book[0:4:2] | Books at index 0, 2. |
| $..book[?(@.isbn)] | All books with an ISBN number. |
| $..book[?(@.price<10)] | All books cheaper than $10. |
| '$..book[?(@.price < 10)]' | All books cheaper than $10. (The path must be quoted if it contains white spaces.) |
| '$..book[?(@["price"] < 10)]' | All books cheaper than $10. |

| Path | Description |
| --- | --- |
| '$..book[?(@.["price"] < 10)]' | All books cheaper than $10. |
| $..book[?(@.price>=10&&@.price<=100)] | All books in the price range of $10 to $100, inclusive. |
| '$..book[?(@.price>=10 && @.price<=100)]' | All books in the price range of $10 to $100, inclusive. (The path must be quoted if it contains white spaces.) |
| $..book[?(@.sold==true\|\|@.in-stock==false)] | All books sold or out of stock. |
| '$..book[?(@.sold == true \|\| @.in-stock == false)]' | All books sold or out of stock. (The path must be quoted if it contains white spaces.) |
| '$.store.book[?(@.["category"] == "fiction")]' | All books in the fiction category. |
| '$.store.book[?(@.["category"] != "fiction")]' | All books in nonfiction categories. |

Additional filter expression examples:

```
127.0.0.1:6379> JSON.SET k1 . '{"books": [{"price":5,"sold":true,"in-
stock":true,"title":"foo"}, {"price":15,"sold":false,"title":"abc"}]}'
OK
127.0.0.1:6379> JSON.GET k1 $.books[?(@.price>1&&@.price<20&&@.in-stock)]
"[{\"price\":5,\"sold\":true,\"in-stock\":true,\"title\":\"foo\"}]"
127.0.0.1:6379> JSON.GET k1 '$.books[?(@.price>1 && @.price<20 && @.in-stock)]'
"[{\"price\":5,\"sold\":true,\"in-stock\":true,\"title\":\"foo\"}]"
127.0.0.1:6379> JSON.GET k1 '$.books[?((@.price>1 && @.price<20) && (@.sold==false))]'
"[{\"price\":15,\"sold\":false,\"title\":\"abc\"}]"
127.0.0.1:6379> JSON.GET k1 '$.books[?(@.title == "abc")]'
[{"price":15,"sold":false,"title":"abc"}]

127.0.0.1:6379> JSON.SET k2 . '[1,2,3,4,5]'
127.0.0.1:6379> JSON.GET k2 $.*.[?(@>2)]
"[3,4,5]"
127.0.0.1:6379> JSON.GET k2 '$.*.[?(@ > 2)]'
"[3,4,5]"

127.0.0.1:6379> JSON.SET k3 . '[true,false,true,false,null,1,2,3,4]'
```

```
OK
127.0.0.1:6379> JSON.GET k3 $.*.[?(@==true)]
"[true,true]"
127.0.0.1:6379> JSON.GET k3 '$.*.[?(@ == true)]'
"[true,true]"
127.0.0.1:6379> JSON.GET k3 $.*.[?(@>1)]
"[2,3,4]"
127.0.0.1:6379> JSON.GET k3 '$.*.[?(@ > 1)]'
"[2,3,4]"
```

### Restricted syntax

| Symbol/Expression | Description |
|---|---|
| . or [] | Child operator. |
| [] | Array subscript operator. Index is 0-based. |

### Examples

| Path | Description |
|---|---|
| .store.book[0].author | The author of the first book. |
| .store.book[-1].author | The author of the last book. |
| .address.city | City name. |
| ["store"]["book"][0]["title"] | The title of the first book. |
| ["store"]["book"][-1]["title"] | The title of the last book. |

**Note**

All Goessner content cited in this documentation is subject to the Creative Commons License.

## Common error prefixes

Each error message has a prefix. The following is a list of common error prefixes.

| Prefix | Description |
| --- | --- |
| ERR | A general error. |
| LIMIT | An error that occurs when the size limit is exceeded. For example, the document size limit or nesting depth limit was exceeded. |
| NONEXISTENT | A key or path does not exist. |
| OUTOFBOUNDARIES | Array index out of bounds. |
| SYNTAXERR | Syntax error. |
| WRONGTYPE | Wrong value type. |

## JSON-related metrics

The following JSON info metrics are provided:

| Info | Description |
| --- | --- |
| json_total_memory_bytes | Total memory allocated to JSON objects. |
| json_num_documents | Total number of documents in Valkey or Redis OSS. |

To query core metrics, run the following command:

```
info json_core_metrics
```

## How ElastiCache for Valkey and Redis OSS interacts with JSON

The following section describes how ElastiCache for Valkey and Redis OSS interacts with the JSON data type.

### Operator precedence

When evaluating conditional expressions for filtering, &&s take precedence first, and then ||s are evaluated, as is common across most languages. Operations inside of parentheses are run first.

### Maximum path nesting limit behavior

The maximum path nesting limit in ElastiCache for Redis OSS is 128. So a value like `$.a.b.c.d...` can only reach 128 levels.

### Handling numeric values

JSON doesn't have separate data types for integers and floating point numbers. They are all called numbers.

Numerical representations:

When a JSON number is received on input, it is converted into one of the two internal binary representations: a 64-bit signed integer or a 64-bit IEEE double precision floating point. The original string and all of its formatting are not retained. Thus, when a number is output as part of a JSON response, it is converted from the internal binary representation to a printable string that uses generic formatting rules. These rules might result in a different string being generated than was received.

Arithmetic commands `NUMINCRBY` and `NUMMULTBY`:

- If both numbers are integers and the result is out of the range of `int64`, it automatically becomes a 64-bit IEEE double precision floating point number.
- If at least one of the numbers is a floating point, the result is a 64-bit IEEE double precision floating point number.
- If the result exceeds the range of 64-bit IEEE double, the command returns an `OVERFLOW` error.

For a detailed list of available commands, see Supported Valkey and Redis OSS commands.

### Direct array filtering

ElastiCache for Valkey or Redis OSS filters array objects directly.

For data like `[0,1,2,3,4,5,6]` and a path query like `$[?(@<4)]`, or data like `{"my_key": [0,1,2,3,4,5,6]}` and a path query like `$.my_key[?(@<4)]`, ElastiCache would return [0,1,2,3] in both circumstances.

### Array indexing behavior

ElastiCache for Valkey or Redis OSS allows both positive and negative indexes for arrays. For an array of length five, 0 would query the first element, 1 the second, and so on. Negative numbers start at the end of the array, so -1 would query the fifth element, -2 the fourth element, and so on.

To ensure predictable behavior for customers, ElastiCache does not round array indexes down or up, so if you have an array with a length of 5, calling index 5 or higher, or -6 or lower, would not produce a result.

### Strict syntax evaluation

ElastiCache does not allow JSON paths with invalid syntax, even if a subset of the path contains a valid path. This is to maintain correct behavior for our customers.

## Supported Valkey and Redis OSS commands

ElastiCache supports the following Valkey and Redis OSS JSON commands:

### Topics


- JSON.ARRAPPEND
- JSON.ARRINDEX
- JSON.ARRINSERT
- JSON.ARRLEN
- JSON.ARRPOP
- JSON.ARRTRIM
- JSON.CLEAR
- JSON.DEBUG
- JSON.DEL

- JSON.FORGET
- JSON.GET
- JSON.MGET
- JSON.MSET
- JSON.NUMINCRBY
- JSON.NUMMULTBY
- JSON.OBJLEN
- JSON.OBJKEYS
- JSON.RESP
- JSON.SET
- JSON.STRAPPEND
- JSON.STRLEN
- JSON.TOGGLE
- JSON.TYPE

## JSON.ARRAPPEND

Appends one or more values to the array values at the path.

Syntax

```
JSON.ARRAPPEND <key> <path> <json> [json ...]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path.
- json (required) – The JSON value to be appended to the array.

### Return

If the path is enhanced syntax:

- Array of integers that represent the new length of the array at each path.
- If a value is not an array, its corresponding return value is null.

- NONEXISTENT error if the path does not exist.

If the path is restricted syntax:

- Integer, the array's new length.
- If multiple array values are selected, the command returns the new length of the first updated array.
- WRONGTYPE error if the value at the path is not an array.
- SYNTAXERR error if one of the input json arguments is not a valid JSON string.
- NONEXISTENT error if the path does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRAPPEND  k1 $[*] '"c"'
1) (integer) 1
2) (integer) 2
3) (integer) 3
127.0.0.1:6379> JSON.GET k1
"[[\"c\"],[\"a\",\"c\"],[\"a\",\"b\",\"c\"]]"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRAPPEND  k1 [-1] '"c"'
(integer) 3
127.0.0.1:6379> JSON.GET k1
"[[],[\"a\"],[\"a\",\"b\",\"c\"]]"
```

## JSON.ARRINDEX

Searches for the first occurrence of a scalar JSON value in the arrays at the path.

- Out of range errors are treated by rounding the index to the array's start and end.
- If start > end, return -1 (not found).

Syntax

```
JSON.ARRINDEX <key> <path> <json-scalar> [start [end]]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path.
- json-scalar (required) – The scalar value to search for. JSON scalar refers to values that are not objects or arrays. That is, string, number, Boolean, and null are scalar values.
- start (optional) – The start index, inclusive. Defaults to 0 if not provided.
- end (optional) – The end index, exclusive. Defaults to 0 if not provided, which means that the last element is included. 0 or -1 means the last element is included.

**Return**

If the path is enhanced syntax:

- Array of integers. Each value is the index of the matching element in the array at the path. The value is -1 if not found.
- If a value is not an array, its corresponding return value is null.

If the path is restricted syntax:

- Integer, the index of matching element, or -1 if not found.
- `WRONGTYPE` error if the value at the path is not an array.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"], ["a", "b", "c"]]'
OK
127.0.0.1:6379> JSON.ARRINDEX k1 $[*] '"b"'
1) (integer) -1
```

```
2) (integer) -1
3) (integer) 1
4) (integer) 1
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"children": ["John", "Jack", "Tom", "Bob", "Mike"]}'
OK
127.0.0.1:6379> JSON.ARRINDEX k1 .children '"Tom"'
(integer) 2
```

## JSON.ARRINSERT

Inserts one or more values into the array values at the path before the index.

Syntax

```
JSON.ARRINSERT <key> <path> <index> <json> [json ...]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path.
- index (required) – An array index before which values are inserted.
- json (required) – The JSON value to be appended to the array.

### Return

If the path is enhanced syntax:

- Array of integers that represent the new length of the array at each path.
- If a value is an empty array, its corresponding return value is null.
- If a value is not an array, its corresponding return value is null.
- `OUTOFBOUNDARIES` error if the index argument is out of bounds.

If the path is restricted syntax:

- Integer, the new length of the array.

- WRONGTYPE error if the value at the path is not an array.
- OUTOFBOUNDARIES error if the index argument is out of bounds.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRINSERT k1 $[*] 0 '"c"'
1) (integer) 1
2) (integer) 2
3) (integer) 3
127.0.0.1:6379> JSON.GET k1
"[[\"c\"],[\"c\",\"a\"],[\"c\",\"a\",\"b\"]]"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRINSERT k1 . 0 '"c"'
(integer) 4
127.0.0.1:6379> JSON.GET k1
"[\"c\",[],[\"a\"],[\"a\",\"b\"]]"
```

## JSON.ARRLEN

Gets the length of the array values at the path.

Syntax

```
JSON.ARRLEN <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

### Return

If the path is enhanced syntax:

- Array of integers that represent the array length at each path.
- If a value is not an array, its corresponding return value is null.
- Null if the document key does not exist.

If the path is restricted syntax:

- Integer, array length.
- If multiple objects are selected, the command returns the first array's length.
- `WRONGTYPE` error if the value at the path is not an array.
- `NONEXISTENT JSON` error if the path does not exist.
- Null if the document key does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"], ["a", "b", "c"]]'
OK
127.0.0.1:6379> JSON.ARRLEN k1 $[*]
1) (integer) 0
2) (integer) 1
3) (integer) 2
4) (integer) 3

127.0.0.1:6379> JSON.SET k2 . '[[], "a", ["a", "b"], ["a", "b", "c"], 4]'
OK
127.0.0.1:6379> JSON.ARRLEN k2 $[*]
1) (integer) 0
2) (nil)
3) (integer) 2
4) (integer) 3
5) (nil)
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"], ["a", "b", "c"]]'
```

```
OK
127.0.0.1:6379> JSON.ARRLEN k1 [*]
(integer) 0
127.0.0.1:6379> JSON.ARRLEN k1 [1]
(integer) 1
127.0.0.1:6379> JSON.ARRLEN k1 [2]
(integer) 2

127.0.0.1:6379> JSON.SET k2 . '[[], "a", ["a", "b"], ["a", "b", "c"], 4]'
OK
127.0.0.1:6379> JSON.ARRLEN k2 [1]
(error) WRONGTYPE JSON element is not an array
127.0.0.1:6379> JSON.ARRLEN k2 [0]
(integer) 0
127.0.0.1:6379> JSON.ARRLEN k2 [6]
(error) OUTOFBOUNDARIES Array index is out of bounds
127.0.0.1:6379> JSON.ARRLEN k2 a.b
(error) NONEXISTENT JSON path does not exist
```

## JSON.ARRPOP

Removes and returns element at the index from the array. Popping an empty array returns null.

Syntax

```
JSON.ARRPOP <key> [path [index]]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.
- index (optional) – The position in the array to start popping from.
  - Defaults to -1 if not provided, which means the last element.
  - Negative value means position from the last element.
  - Out of boundary indexes are rounded to their respective array boundaries.

### Return

If the path is enhanced syntax:

- Array of bulk strings that represent popped values at each path.

- If a value is an empty array, its corresponding return value is null.
- If a value is not an array, its corresponding return value is null.

If the path is restricted syntax:

- Bulk string, which represents the popped JSON value.
- Null if the array is empty.
- WRONGTYPE error if the value at the path is not an array.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRPOP k1 $[*]
1) (nil)
2) "\"a\""
3) "\"b\""
127.0.0.1:6379> JSON.GET k1
"[[],[],[\"a\"]]"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRPOP k1
"[\"a\",\"b\"]"
127.0.0.1:6379> JSON.GET k1
"[[],[\"a\"]]"

127.0.0.1:6379> JSON.SET k2 . '[[], ["a"], ["a", "b"]]'
OK
127.0.0.1:6379> JSON.ARRPOP k2 . 0
"[]"
127.0.0.1:6379> JSON.GET k2
"[[\"a\"],[\"a\",\"b\"]]"
```

## JSON.ARRTRIM

Trims an arrays at the path so that it becomes a subarray [start, end], both inclusive.

- If the array is empty, do nothing, return 0.
- If start <0, treat it as 0.
- If end >= size (size of the array), treat it as size-1.
- If start >= size or start > end, empty the array and return 0.

Syntax

```
JSON.ARRTRIM <key> <path> <start> <end>
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path.
- start (required) – The start index, inclusive.
- end (required) – The end index, inclusive.

### Return

If the path is enhanced syntax:

- Array of integers that represent the new length of the array at each path.
- If a value is an empty array, its corresponding return value is null.
- If a value is not an array, its corresponding return value is null.
- `OUTOFBOUNDARIES` error if an index argument is out of bounds.

If the path is restricted syntax:

- Integer, the new length of the array.
- Null if the array is empty.
- `WRONGTYPE` error if the value at the path is not an array.
- `OUTOFBOUNDARIES` error if an index argument is out of bounds.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[[], ["a"], ["a", "b"], ["a", "b", "c"]]'
OK
127.0.0.1:6379> JSON.ARRTRIM k1 $[*] 0 1
1) (integer) 0
2) (integer) 1
3) (integer) 2
4) (integer) 2
127.0.0.1:6379> JSON.GET k1
"[[],[\"a\"],[\"a\",\"b\"],[\"a\",\"b\"]]"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"children": ["John", "Jack", "Tom", "Bob", "Mike"]}'
OK
127.0.0.1:6379> JSON.ARRTRIM k1 .children 0 1
(integer) 2
127.0.0.1:6379> JSON.GET k1 .children
"[\"John\",\"Jack\"]"
```

## JSON.CLEAR

Clears the arrays or an object at the path.

Syntax

```
JSON.CLEAR <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

### Return

- Integer, the number of containers cleared.
- Clearing an empty array or object accounts for 1 container cleared.
- Clearing a non-container value returns 0.

**Examples**

```
127.0.0.1:6379> JSON.SET k1 . '[[], [0], [0,1], [0,1,2], 1, true, null, "d"]'
OK
127.0.0.1:6379>  JSON.CLEAR k1  $[*]
(integer) 7
127.0.0.1:6379> JSON.CLEAR k1  $[*]
(integer) 4
127.0.0.1:6379> JSON.SET k2 . '{"children": ["John", "Jack", "Tom", "Bob", "Mike"]}'
OK
127.0.0.1:6379> JSON.CLEAR k2 .children
(integer) 1
127.0.0.1:6379> JSON.GET k2 .children
"[]"
```

## JSON.DEBUG

Reports information. Supported subcommands are:

- MEMORY <key> [path] – Reports memory usage in bytes of a JSON value. Path defaults to the root if not provided.
- FIELDS <key> [path] – Reports the number of fields at the specified document path. Path defaults to the root if not provided. Each non-container JSON value counts as one field. Objects and arrays recursively count one field for each of their containing JSON values. Each container value, except the root container, counts as one additional field.
- HELP – Prints help messages of the command.

Syntax

```
JSON.DEBUG <subcommand & arguments>
```

Depends on the subcommand:

MEMORY

- If the path is enhanced syntax:
  - Returns an array of integers that represent memory size (in bytes) of JSON value at each path.
  - Returns an empty array if the Valkey or Redis OSS key does not exist.

- If the path is restricted syntax:
  - Returns an integer, memory size, and the JSON value in bytes.
  - Returns null if the Valkey or Redis OSS key does not exist.

FIELDS

- If the path is enhanced syntax:
  - Returns an array of integers that represent the number of fields of JSON value at each path.
  - Returns an empty array if the Valkey or Redis OSS key does not exist.
- If the path is restricted syntax:
  - Returns an integer, number of fields of the JSON value.
  - Returns null if the Valkey or Redis OSS key does not exist.

HELP – Returns an array of help messages.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[1, 2.3, "foo", true, null, {}, [], {"a":1, "b":2},
 [1,2,3]]'
OK
127.0.0.1:6379> JSON.DEBUG MEMORY k1 $[*]
1) (integer) 16
2) (integer) 16
3) (integer) 19
4) (integer) 16
5) (integer) 16
6) (integer) 16
7) (integer) 16
8) (integer) 50
9) (integer) 64
127.0.0.1:6379> JSON.DEBUG FIELDS k1 $[*]
1) (integer) 1
2) (integer) 1
3) (integer) 1
4) (integer) 1
5) (integer) 1
6) (integer) 0
```

```
7) (integer) 0
8) (integer) 2
9) (integer) 3
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 .
 '{"firstName":"John","lastName":"Smith","age":27,"weight":135.25,"isAlive":true,"address":
{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021-3100"},"phoneNumbers":
[{"type":"home","number":"212 555-1234"},{"type":"office","number":"646
 555-4567"}],"children":[],"spouse":null}'
OK
127.0.0.1:6379> JSON.DEBUG MEMORY k1
(integer) 632
127.0.0.1:6379> JSON.DEBUG MEMORY k1 .phoneNumbers
(integer) 166

127.0.0.1:6379> JSON.DEBUG FIELDS k1
(integer) 19
127.0.0.1:6379> JSON.DEBUG FIELDS k1 .address
(integer) 4

127.0.0.1:6379> JSON.DEBUG HELP
1) JSON.DEBUG MEMORY <key> [path] - report memory size (bytes) of the JSON element.
 Path defaults to root if not provided.
2) JSON.DEBUG FIELDS <key> [path] - report number of fields in the JSON element. Path
 defaults to root if not provided.
3) JSON.DEBUG HELP - print help message.
```

## JSON.DEL

Deletes the JSON values at the path in a document key. If the path is the root, it is equivalent to deleting the key from Valkey or Redis OSS.

Syntax

```
JSON.DEL <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

**Return**

- Number of elements deleted.
- 0 if the Valkey or Redis OSS key does not exist.
- 0 if the JSON path is invalid or does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}, "d":{"a":1,
 "b":2, "c":3}, "e": [1,2,3,4,5]}'
OK
127.0.0.1:6379> JSON.DEL k1 $.d.*
(integer) 3
127.0.0.1:6379> JSOn.GET k1
"{\"a\":{},\"b\":{\"a\":1},\"c\":{\"a\":1,\"b\":2},\"d\":{},\"e\":[1,2,3,4,5]}"
127.0.0.1:6379> JSON.DEL k1 $.e[*]
(integer) 5
127.0.0.1:6379> JSOn.GET k1
"{\"a\":{},\"b\":{\"a\":1},\"c\":{\"a\":1,\"b\":2},\"d\":{},\"e\":[]}"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}, "d":{"a":1,
 "b":2, "c":3}, "e": [1,2,3,4,5]}'
OK
127.0.0.1:6379> JSON.DEL k1 .d.*
(integer) 3
127.0.0.1:6379> JSON.GET k1
"{\"a\":{},\"b\":{\"a\":1},\"c\":{\"a\":1,\"b\":2},\"d\":{},\"e\":[1,2,3,4,5]}"
127.0.0.1:6379> JSON.DEL k1 .e[*]
(integer) 5
127.0.0.1:6379> JSON.GET k1
"{\"a\":{},\"b\":{\"a\":1},\"c\":{\"a\":1,\"b\":2},\"d\":{},\"e\":[]}"
```

## JSON.FORGET

An alias of JSON.DEL.

## JSON.GET

Returns the serialized JSON at one or multiple paths.

Syntax

```
JSON.GET <key>
[INDENT indentation-string]
[NEWLINE newline-string]
[SPACE space-string]
[NOESCAPE]
[path ...]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- INDENT/NEWLINE/SPACE (optional) – Controls the format of the returned JSON string, that is, "pretty print". The default value of each one is an empty string. They can be overridden in any combination. They can be specified in any order.
- NOESCAPE - Optional, allowed to be present for legacy compatibility and has no other effect.
- path (optional) – Zero or more JSON paths, defaults to the root if none is given. The path arguments must be placed at the end.

### Return

Enhanced path syntax:

If one path is given:

- Returns serialized string of an array of values.
- If no value is selected, the command returns an empty array.

If multiple paths are given:

- Returns a stringified JSON object, in which each path is a key.
- If there are mixed enhanced and restricted path syntax, the result conforms to the enhanced syntax.

- If a path does not exist, its corresponding value is an empty array.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 .
 '{"firstName":"John","lastName":"Smith","age":27,"weight":135.25,"isAlive":true,"address":
{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021-3100"},"phoneNumbers":
[{"type":"home","number":"212 555-1234"},{"type":"office","number":"646
 555-4567"}],"children":[],"spouse":null}'
OK
127.0.0.1:6379> JSON.GET k1 $.address.*
"[\"21 2nd Street\",\"New York\",\"NY\",\"10021-3100\"]"
127.0.0.1:6379> JSON.GET k1 indent "\t" space " " NEWLINE "\n" $.address.*
"[\n\t\"21 2nd Street\",\n\t\"New York\",\n\t\"NY\",\n\t\"10021-3100\"\n]"
127.0.0.1:6379> JSON.GET k1 $.firstName $.lastName $.age
"{\"$.firstName\":[\"John\"],\"$.lastName\":[\"Smith\"],\"$.age\":[27]}"
127.0.0.1:6379> JSON.SET k2 . '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}}'
OK
127.0.0.1:6379> json.get k2 $..*
"[{},{\"a\":1},{\"a\":1,\"b\":2},1,1,2]"
```

Restricted path syntax:

```
 127.0.0.1:6379> JSON.SET k1 .
 '{"firstName":"John","lastName":"Smith","age":27,"weight":135.25,"isAlive":true,"address":
{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021-3100"},"phoneNumbers":
[{"type":"home","number":"212 555-1234"},{"type":"office","number":"646
 555-4567"}],"children":[],"spouse":null}'
OK
127.0.0.1:6379> JSON.GET k1 .address
"{\"street\":\"21 2nd Street\",\"city\":\"New York\",\"state\":\"NY\",\"zipcode\":
\"10021-3100\"}"
127.0.0.1:6379> JSON.GET k1 indent "\t" space " " NEWLINE "\n" .address
"{\n\t\"street\": \"21 2nd Street\",\n\t\"city\": \"New York\",\n\t\"state\": \"NY\",\n
\t\"zipcode\": \"10021-3100\"\n}"
127.0.0.1:6379> JSON.GET k1 .firstName .lastName .age
"{\".firstName\":\"John\",\".lastName\":\"Smith\",\".age\":27}"
```

## JSON.MGET

Gets serialized JSONs at the path from multiple document keys. It returns null for a nonexistent key or JSON path.

### Syntax

```
JSON.MGET <key> [key ...] <path>
```

- key (required) – One or more Valkey or Redis OSS keys of document type.
- path (required) – A JSON path.

### Return

- Array of bulk strings. The size of the array is equal to the number of keys in the command. Each element of the array is populated with either (a) the serialized JSON as located by the path or (b) null if the key does not exist, the path does not exist in the document, or the path is invalid (syntax error).
- If any of the specified keys exists and is not a JSON key, the command returns `WRONGTYPE` error.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"address":{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021"}}'
OK
127.0.0.1:6379> JSON.SET k2 . '{"address":{"street":"5 main
 Street","city":"Boston","state":"MA","zipcode":"02101"}}'
OK
127.0.0.1:6379> JSON.SET k3 . '{"address":{"street":"100 Park
 Ave","city":"Seattle","state":"WA","zipcode":"98102"}}'
OK
127.0.0.1:6379> JSON.MGET k1 k2 k3 $.address.city
1) "[\"New York\"]"
2) "[\"Boston\"]"
3) "[\"Seattle\"]"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"address":{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021"}}'
OK
127.0.0.1:6379> JSON.SET k2 . '{"address":{"street":"5 main
 Street","city":"Boston","state":"MA","zipcode":"02101"}}'
OK
127.0.0.1:6379> JSON.SET k3 . '{"address":{"street":"100 Park
 Ave","city":"Seattle","state":"WA","zipcode":"98102"}}'
OK

127.0.0.1:6379> JSON.MGET k1 k2 k3 .address.city
1) "\"New York\""
2) "\"Seattle\""
3) "\"Seattle\""
```

## JSON.MSET

Supported for Valkey version 8.1 and above.

Set JSON values for multiple keys. The operation is atomic. Either all values are set or none is set.

### Syntax

```
JSON.MSET key path json [ key path json ... ]
```

- If the path calls for an object member:
  - If the parent element does not exist, the command will return NONEXISTENT error.
  - If the parent element exists but is not an object, the command will return ERROR.
  - If the parent element exists and is an object:
    - If the member does not exist, a new member will be appended to the parent object if and only if the parent object is the last child in the path. Otherwise, the command will return NONEXISTENT error.
    - If the member exists, its value will be replaced by the JSON value.
- If the path calls for an array index:

- If the parent element does not exist, the command will return a NONEXISTENT error.
- If the parent element exists but is not an array, the command will return ERROR.
- If the parent element exists but the index is out of bounds, the command will return OUTOFBOUNDARIES error.
- If the parent element exists and the index is valid, the element will be replaced by the new JSON value.

- If the path calls for an object or array, the value (object or array) will be replaced by the new JSON value.

**Return**

- Simple string reply: 'OK' if the operation was successful.
- Simple error reply: If the operation failed.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.MSET k1 . '[1,2,3,4,5]' k2 . '{"a":{"a":1, "b":2, "c":3}}' k3 .
 '{"a": [1,2,3,4,5]}'
OK
127.0.0.1:6379> JSON.GET k1
"[1,2,3,4,5]"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{\"a\":1,\"b\":2,\"c\":3}}"
127.0.0.1:6379> JSON.MSET k2 $.a.* '0' k3 $.a[*] '0'
OK
127.0.0.1:6379> JSON.GET k2
"{\"a\":{\"a\":0,\"b\":0,\"c\":0}}"
127.0.0.1:6379> JSON.GET k3
"{\"a\":[0,0,0,0,0]}"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.MSET k1 . '{"name": "John","address": {"street": "123 Main
 St","city": "Springfield"},"phones": ["555-1234","555-5678"]}'
OK
127.0.0.1:6379> JSON.MSET k1 .address.street '"21 2nd Street"' k1 .address.city '"New
 York"'
```

```
OK
127.0.0.1:6379> JSON.GET k1 .address.street
"\"21 2nd Street\""
127.0.0.1:6379> JSON.GET k1 .address.city
"\"New York\""
```

## JSON.NUMINCRBY

Increments the number values at the path by a given number.

Syntax

```
JSON.NUMINCRBY <key> <path> <number>
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path.
- number (required) – A number.

**Return**

If the path is enhanced syntax:

- Array of bulk strings that represents the resulting value at each path.
- If a value is not a number, its corresponding return value is null.
- `WRONGTYPE` error if the number cannot be parsed.
- `OVERFLOW` error if the result is out of the range of 64-bit IEEE double.
- `NONEXISTENT` if the document key does not exist.

If the path is restricted syntax:

- Bulk string that represents the resulting value.
- If multiple values are selected, the command returns the result of the last updated value.
- `WRONGTYPE` error if the value at the path is not a number.
- `WRONGTYPE` error if the number cannot be parsed.
- `OVERFLOW` error if the result is out of the range of 64-bit IEEE double.
- `NONEXISTENT` if the document key does not exist.

## Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k1 $.d[*] 10
"[11,12,13]"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[1],\"c\":[1,2],\"d\":[11,12,13]}"

127.0.0.1:6379> JSON.SET k1 $ '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k1 $.a[*] 1
"[]"
127.0.0.1:6379> JSON.NUMINCRBY k1 $.b[*] 1
"[2]"
127.0.0.1:6379> JSON.NUMINCRBY k1 $.c[*] 1
"[2,3]"
127.0.0.1:6379> JSON.NUMINCRBY k1 $.d[*] 1
"[2,3,4]"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[2,3],\"d\":[2,3,4]}"

127.0.0.1:6379> JSON.SET k2 $ '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}, "d":{"a":1,
 "b":2, "c":3}}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k2 $.a.* 1
"[]"
127.0.0.1:6379> JSON.NUMINCRBY k2 $.b.* 1
"[2]"
127.0.0.1:6379> JSON.NUMINCRBY k2 $.c.* 1
"[2,3]"
127.0.0.1:6379> JSON.NUMINCRBY k2 $.d.* 1
"[2,3,4]"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":2,\"b\":3},\"d\":{\"a\":2,\"b\":3,\"c\":4}}"

127.0.0.1:6379> JSON.SET k3 $ '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"b"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k3 $.a.* 1
"[null]"
127.0.0.1:6379> JSON.NUMINCRBY k3 $.b.* 1
```

```
"[null,2]"
127.0.0.1:6379> JSON.NUMINCRBY k3 $.c.* 1
"[null,null]"
127.0.0.1:6379> JSON.NUMINCRBY k3 $.d.* 1
"[2,null,4]"
127.0.0.1:6379> JSON.GET k3
"{\"a\":{\"a\":\"a\"},\"b\":{\"a\":\"a\",\"b\":2},\"c\":{\"a\":\"a\",\"b\":\"b\"},\"d
\":{\"a\":2,\"b\":\"b\",\"c\":4}}"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k1 .d[1] 10
"12"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[1],\"c\":[1,2],\"d\":[1,12,3]}"

127.0.0.1:6379> JSON.SET k1 . '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k1 .a[*] 1
(error) NONEXISTENT JSON path does not exist
127.0.0.1:6379> JSON.NUMINCRBY k1 .b[*] 1
"2"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[1,2],\"d\":[1,2,3]}"
127.0.0.1:6379> JSON.NUMINCRBY k1 .c[*] 1
"3"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[2,3],\"d\":[1,2,3]}"
127.0.0.1:6379> JSON.NUMINCRBY k1 .d[*] 1
"4"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[2,3],\"d\":[2,3,4]}"

127.0.0.1:6379> JSON.SET k2 . '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}, "d":{"a":1,
 "b":2, "c":3}}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k2 .a.* 1
(error) NONEXISTENT JSON path does not exist
127.0.0.1:6379> JSON.NUMINCRBY k2 .b.* 1
"2"
```

```
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":1,\"b\":2},\"d\":{\"a\":1,\"b\":2,\"c\":3}}"
127.0.0.1:6379> JSON.NUMINCRBY k2 .c.* 1
"3"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":2,\"b\":3},\"d\":{\"a\":1,\"b\":2,\"c\":3}}"
127.0.0.1:6379> JSON.NUMINCRBY k2 .d.* 1
"4"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":2,\"b\":3},\"d\":{\"a\":2,\"b\":3,\"c\":4}}"

127.0.0.1:6379> JSON.SET k3 . '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"b"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.NUMINCRBY k3 .a.* 1
(error) WRONGTYPE JSON element is not a number
127.0.0.1:6379> JSON.NUMINCRBY k3 .b.* 1
"2"
127.0.0.1:6379> JSON.NUMINCRBY k3 .c.* 1
(error) WRONGTYPE JSON element is not a number
127.0.0.1:6379> JSON.NUMINCRBY k3 .d.* 1
"4"
```

## JSON.NUMMULTBY

Multiplies the number values at the path by a given number.

Syntax

```
JSON.NUMMULTBY <key> <path> <number>
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path.
- number (required) – A number.

### Return

If the path is enhanced syntax:

- Array of bulk strings that represent the resulting value at each path.

- If a value is not a number, its corresponding return value is null.
- WRONGTYPE error if the number cannot be parsed.
- OVERFLOW error if the result is out of the range of a 64-bit IEEE double precision floating point number.
- NONEXISTENT if the document key does not exist.

If the path is restricted syntax:

- Bulk string that represents the resulting value.
- If multiple values are selected, the command returns the result of the last updated value.
- WRONGTYPE error if the value at the path is not a number.
- WRONGTYPE error if the number cannot be parsed.
- OVERFLOW error if the result is out of the range of a 64-bit IEEE double.
- NONEXISTENT if the document key does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k1 $.d[*] 2
"[2,4,6]"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[1],\"c\":[1,2],\"d\":[2,4,6]}"

127.0.0.1:6379> JSON.SET k1 $ '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k1 $.a[*] 2
"[]"
127.0.0.1:6379> JSON.NUMMULTBY k1 $.b[*] 2
"[2]"
127.0.0.1:6379> JSON.NUMMULTBY k1 $.c[*] 2
"[2,4]"
127.0.0.1:6379> JSON.NUMMULTBY k1 $.d[*] 2
"[2,4,6]"
```

```
127.0.0.1:6379> JSON.SET k2 $ '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}, "d":{"a":1,
 "b":2, "c":3}}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k2 $.a.* 2
"[]"
127.0.0.1:6379> JSON.NUMMULTBY k2 $.b.* 2
"[2]"
127.0.0.1:6379> JSON.NUMMULTBY k2 $.c.* 2
"[2,4]"
127.0.0.1:6379> JSON.NUMMULTBY k2 $.d.* 2
"[2,4,6]"

127.0.0.1:6379> JSON.SET k3 $ '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"b"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k3 $.a.* 2
"[null]"
127.0.0.1:6379> JSON.NUMMULTBY k3 $.b.* 2
"[null,2]"
127.0.0.1:6379> JSON.NUMMULTBY k3 $.c.* 2
"[null,null]"
127.0.0.1:6379> JSON.NUMMULTBY k3 $.d.* 2
"[2,null,6]"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k1 .d[1] 2
"4"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[1],\"c\":[1,2],\"d\":[1,4,3]}"

127.0.0.1:6379> JSON.SET k1 . '{"a":[], "b":[1], "c":[1,2], "d":[1,2,3]}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k1 .a[*] 2
(error) NONEXISTENT JSON path does not exist
127.0.0.1:6379> JSON.NUMMULTBY k1 .b[*] 2
"2"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[1,2],\"d\":[1,2,3]}"
127.0.0.1:6379> JSON.NUMMULTBY k1 .c[*] 2
```

```
"4"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[2,4],\"d\":[1,2,3]}"
127.0.0.1:6379> JSON.NUMMULTBY k1 .d[*] 2
"6"
127.0.0.1:6379> JSON.GET k1
"{\"a\":[],\"b\":[2],\"c\":[2,4],\"d\":[2,4,6]}"

127.0.0.1:6379> JSON.SET k2 . '{"a":{}, "b":{"a":1}, "c":{"a":1, "b":2}, "d":{"a":1,
 "b":2, "c":3}}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k2 .a.* 2
(error) NONEXISTENT JSON path does not exist
127.0.0.1:6379> JSON.NUMMULTBY k2 .b.* 2
"2"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":1,\"b\":2},\"d\":{\"a\":1,\"b\":2,\"c\":3}}"
127.0.0.1:6379> JSON.NUMMULTBY k2 .c.* 2
"4"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":2,\"b\":4},\"d\":{\"a\":1,\"b\":2,\"c\":3}}"
127.0.0.1:6379> JSON.NUMMULTBY k2 .d.* 2
"6"
127.0.0.1:6379> JSON.GET k2
"{\"a\":{},\"b\":{\"a\":2},\"c\":{\"a\":2,\"b\":4},\"d\":{\"a\":2,\"b\":4,\"c\":6}}"

127.0.0.1:6379> JSON.SET k3 . '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"b"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.NUMMULTBY k3 .a.* 2
(error) WRONGTYPE JSON element is not a number
127.0.0.1:6379> JSON.NUMMULTBY k3 .b.* 2
"2"
127.0.0.1:6379> JSON.GET k3
"{\"a\":{\"a\":\"a\"},\"b\":{\"a\":\"a\",\"b\":2},\"c\":{\"a\":\"a\",\"b\":\"b\"},\"d
\":{\"a\":1,\"b\":\"b\",\"c\":3}}"
127.0.0.1:6379> JSON.NUMMULTBY k3 .c.* 2
(error) WRONGTYPE JSON element is not a number
127.0.0.1:6379> JSON.NUMMULTBY k3 .d.* 2
"6"
127.0.0.1:6379> JSON.GET k3
"{\"a\":{\"a\":\"a\"},\"b\":{\"a\":\"a\",\"b\":2},\"c\":{\"a\":\"a\",\"b\":\"b\"},\"d
\":{\"a\":2,\"b\":\"b\",\"c\":6}}"
```

## JSON.OBJLEN

Gets the number of keys in the object values at the path.

Syntax

```
JSON.OBJLEN <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

### Return

If the path is enhanced syntax:

- Array of integers that represent the object length at each path.
- If a value is not an object, its corresponding return value is null.
- Null if the document key does not exist.

If the path is restricted syntax:

- Integer, number of keys in the object.
- If multiple objects are selected, the command returns the first object's length.
- `WRONGTYPE` error if the value at the path is not an object.
- `NONEXISTENT JSON` error if the path does not exist.
- Null if the document key does not exist.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 $ '{"a":{}, "b":{"a":"a"}, "c":{"a":"a", "b":"bb"}, "d":
{"a":1, "b":"b", "c":{"a":3,"b":4}}, "e":1}'
OK
127.0.0.1:6379> JSON.OBJLEN k1 $.a
```

```
1) (integer) 0
127.0.0.1:6379> JSON.OBJLEN k1 $.a.*
(empty array)
127.0.0.1:6379> JSON.OBJLEN k1 $.b
1) (integer) 1
127.0.0.1:6379> JSON.OBJLEN k1 $.b.*
1) (nil)
127.0.0.1:6379> JSON.OBJLEN k1 $.c
1) (integer) 2
127.0.0.1:6379> JSON.OBJLEN k1 $.c.*
1) (nil)
2) (nil)
127.0.0.1:6379> JSON.OBJLEN k1 $.d
1) (integer) 3
127.0.0.1:6379> JSON.OBJLEN k1 $.d.*
1) (nil)
2) (nil)
3) (integer) 2
127.0.0.1:6379> JSON.OBJLEN k1 $.*
1) (integer) 0
2) (integer) 1
3) (integer) 2
4) (integer) 3
5) (nil)
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":{}, "b":{"a":"a"}, "c":{"a":"a", "b":"bb"}, "d":
{"a":1, "b":"b", "c":{"a":3,"b":4}}, "e":1}'
OK
127.0.0.1:6379> JSON.OBJLEN k1 .a
(integer) 0
127.0.0.1:6379> JSON.OBJLEN k1 .a.*
(error) NONEXISTENT JSON path does not exist
127.0.0.1:6379> JSON.OBJLEN k1 .b
(integer) 1
127.0.0.1:6379> JSON.OBJLEN k1 .b.*
(error) WRONGTYPE JSON element is not an object
127.0.0.1:6379> JSON.OBJLEN k1 .c
(integer) 2
127.0.0.1:6379> JSON.OBJLEN k1 .c.*
(error) WRONGTYPE JSON element is not an object
```

```
127.0.0.1:6379> JSON.OBJLEN k1 .d
(integer) 3
127.0.0.1:6379> JSON.OBJLEN k1 .d.*
(integer) 2
127.0.0.1:6379> JSON.OBJLEN k1 .*
(integer) 0
```

## JSON.OBJKEYS

Gets key names in the object values at the path.

Syntax

```
JSON.OBJKEYS <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

### Return

If the path is enhanced syntax:

- Array of array of bulk strings. Each element is an array of keys in a matching object.
- If a value is not an object, its corresponding return value is empty value.
- Null if the document key does not exist.

If the path is restricted syntax:

- Array of bulk strings. Each element is a key name in the object.
- If multiple objects are selected, the command returns the keys of the first object.
- `WRONGTYPE` error if the value at the path is not an object.
- Null if the document key does not exist.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 $ '{"a":{}, "b":{"a":"a"}, "c":{"a":"a", "b":"bb"}, "d":
{"a":1, "b":"b", "c":{"a":3,"b":4}}, "e":1}'
OK
127.0.0.1:6379> JSON.OBJKEYS k1 $.*
1) (empty array)
2) 1) "a"
3) 1) "a"
   2) "b"
4) 1) "a"
   2) "b"
   3) "c"
5) (empty array)
127.0.0.1:6379> JSON.OBJKEYS k1 $.d
1) 1) "a"
   2) "b"
   3) "c"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 $ '{"a":{}, "b":{"a":"a"}, "c":{"a":"a", "b":"bb"}, "d":
{"a":1, "b":"b", "c":{"a":3,"b":4}}, "e":1}'
OK
127.0.0.1:6379> JSON.OBJKEYS k1 .*
1) "a"
127.0.0.1:6379> JSON.OBJKEYS k1 .d
1) "a"
2) "b"
3) "c"
```

## JSON.RESP

Returns the JSON value at the given path in the Valkey or Redis OSS Serialization Protocol (RESP). If the value is container, the response is a RESP array or nested array.

- JSON null is mapped to the RESP Null Bulk String.
- JSON Boolean values are mapped to the respective RESP Simple Strings.
- Integer numbers are mapped to RESP Integers.
- 64-bit IEEE double floating point numbers are mapped to RESP Bulk Strings.

- JSON strings are mapped to RESP Bulk Strings.
- JSON arrays are represented as RESP Arrays, where the first element is the simple string [, followed by the array's elements.
- JSON objects are represented as RESP Arrays, where the first element is the simple string {, followed by key-value pairs, each of which is a RESP bulk string.

Syntax

```
JSON.RESP <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

**Return**

If the path is enhanced syntax:

- Array of arrays. Each array element represents the RESP form of the value at one path.
- Empty array if the document key does not exist.

If the path is restricted syntax:

- Array that represents the RESP form of the value at the path.
- Null if the document key does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 .
 '{"firstName":"John","lastName":"Smith","age":27,"weight":135.25,"isAlive":true,"address":
{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021-3100"},"phoneNumbers":
[{"type":"home","number":"212 555-1234"},{"type":"office","number":"646
 555-4567"}],"children":[],"spouse":null}'
OK
```

```
127.0.0.1:6379> JSON.RESP k1 $.address
1) 1) {
   2) 1) "street"
      2) "21 2nd Street"
   3) 1) "city"
      2) "New York"
   4) 1) "state"
      2) "NY"
   5) 1) "zipcode"
      2) "10021-3100"

127.0.0.1:6379> JSON.RESP k1 $.address.*
1) "21 2nd Street"
2) "New York"
3) "NY"
4) "10021-3100"

127.0.0.1:6379> JSON.RESP k1 $.phoneNumbers
1) 1) [
   2) 1) {
      2) 1) "type"
         2) "home"
      3) 1) "number"
         2) "555 555-1234"
   3) 1) {
      2) 1) "type"
         2) "office"
      3) 1) "number"
         2) "555 555-4567"

127.0.0.1:6379> JSON.RESP k1 $.phoneNumbers[*]
1) 1) {
   2) 1) "type"
      2) "home"
   3) 1) "number"
      2) "212 555-1234"
2) 1) {
   2) 1) "type"
      2) "office"
   3) 1) "number"
      2) "555 555-4567"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 .
 '{"firstName":"John","lastName":"Smith","age":27,"weight":135.25,"isAlive":true,"address":
{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021-3100"},"phoneNumbers":
[{"type":"home","number":"212 555-1234"},{"type":"office","number":"646
 555-4567"}],"children":[],"spouse":null}'
OK

127.0.0.1:6379> JSON.RESP k1 .address
1) {
2) 1) "street"
   2) "21 2nd Street"
3) 1) "city"
   2) "New York"
4) 1) "state"
   2) "NY"
5) 1) "zipcode"
   2) "10021-3100"

127.0.0.1:6379> JSON.RESP k1
 1) {
 2) 1) "firstName"
    2) "John"
 3) 1) "lastName"
    2) "Smith"
 4) 1) "age"
    2) (integer) 27
 5) 1) "weight"
    2) "135.25"
 6) 1) "isAlive"
    2) true
 7) 1) "address"
    2) 1) {
       2) 1) "street"
          2) "21 2nd Street"
       3) 1) "city"
          2) "New York"
       4) 1) "state"
          2) "NY"
       5) 1) "zipcode"
          2) "10021-3100"
 8) 1) "phoneNumbers"
```

```
    2) 1) [
       2) 1) {
          2) 1) "type"
             2) "home"
          3) 1) "number"
             2) "212 555-1234"
       3) 1) {
          2) 1) "type"
             2) "office"
          3) 1) "number"
             2) "555 555-4567"
 9) 1) "children"
    2) 1) [
10) 1) "spouse"
    2) (nil)
```

## JSON.SET

Sets JSON values at the path.

If the path calls for an object member:

- If the parent element does not exist, the command returns a NONEXISTENT error.
- If the parent element exists but is not an object, the command returns ERROR.
- If the parent element exists and is an object:
  - If the member does not exist, a new member will be appended to the parent object if and only if the parent object is the last child in the path. Otherwise, the command returns a NONEXISTENT error.
  - If the member exists, its value will be replaced by the JSON value.

If the path calls for an array index:

- If the parent element does not exist, the command returns a NONEXISTENT error.
- If the parent element exists but is not an array, the command returns ERROR.
- If the parent element exists but the index is out of bounds, the command returns an OUTOFBOUNDARIES error.
- If the parent element exists and the index is valid, the element will be replaced by the new JSON value.

If the path calls for an object or array, the value (object or array) will be replaced by the new JSON value.

Syntax

```
JSON.SET <key> <path> <json> [NX | XX]
```

[NX | XX] Where you can have 0 or 1 of [NX | XX] identifiers.

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (required) – A JSON path. For a new key, the JSON path must be the root ".".
- NX (optional) – If the path is the root, set the value only if the key does not exist. That is, insert a new document. If the path is not the root, set the value only if the path does not exist. That is, insert a value into the document.
- XX (optional) – If the path is the root, set the value only if the key exists. That is, replace the existing document. If the path is not the root, set the value only if the path exists. That is, update the existing value.

**Return**

- Simple String 'OK' on success.
- Null if the NX or XX condition is not met.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":{"a":1, "b":2, "c":3}}'
OK
127.0.0.1:6379> JSON.SET k1 $.a.* '0'
OK
127.0.0.1:6379> JSON.GET k1
"{\"a\":{\"a\":0,\"b\":0,\"c\":0}}"

127.0.0.1:6379> JSON.SET k2 . '{"a": [1,2,3,4,5]}'
OK
127.0.0.1:6379> JSON.SET k2 $.a[*] '0'
OK
127.0.0.1:6379> JSON.GET k2
```

```
"{\"a\":[0,0,0,0,0]}"
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"c":{"a":1, "b":2}, "e": [1,2,3,4,5]}'
OK
127.0.0.1:6379> JSON.SET k1 .c.a '0'
OK
127.0.0.1:6379> JSON.GET k1
"{\"c\":{\"a\":0,\"b\":2},\"e\":[1,2,3,4,5]}"
127.0.0.1:6379> JSON.SET k1 .e[-1] '0'
OK
127.0.0.1:6379> JSON.GET k1
"{\"c\":{\"a\":0,\"b\":2},\"e\":[1,2,3,4,0]}"
127.0.0.1:6379> JSON.SET k1 .e[5] '0'
(error) OUTOFBOUNDARIES Array index is out of bounds
```

## JSON.STRAPPEND

Appends a string to the JSON strings at the path.

Syntax

```
JSON.STRAPPEND <key> [path] <json_string>
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.
- json_string (required) – The JSON representation of a string. Note that a JSON string must be quoted. For example: '"string example"'.

### Return

If the path is enhanced syntax:

- Array of integers that represent the new length of the string at each path.
- If a value at the path is not a string, its corresponding return value is null.
- SYNTAXERR error if the input json argument is not a valid JSON string.

- NONEXISTENT error if the path does not exist.

If the path is restricted syntax:

- Integer, the string's new length.
- If multiple string values are selected, the command returns the new length of the last updated string.
- WRONGTYPE error if the value at the path is not a string.
- WRONGTYPE error if the input json argument is not a valid JSON string.
- NONEXISTENT error if the path does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 $ '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"bb"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.STRAPPEND k1 $.a.a '"a"'
1) (integer) 2
127.0.0.1:6379> JSON.STRAPPEND k1 $.a.* '"a"'
1) (integer) 3
127.0.0.1:6379> JSON.STRAPPEND k1 $.b.* '"a"'
1) (integer) 2
2) (nil)
127.0.0.1:6379> JSON.STRAPPEND k1 $.c.* '"a"'
1) (integer) 2
2) (integer) 3
127.0.0.1:6379> JSON.STRAPPEND k1 $.c.b '"a"'
1) (integer) 4
127.0.0.1:6379> JSON.STRAPPEND k1 $.d.* '"a"'
1) (nil)
2) (integer) 2
3) (nil)
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"bb"}, "d":{"a":1, "b":"b", "c":3}}'
```

```
OK
127.0.0.1:6379> JSON.STRAPPEND k1 .a.a '"a"'
(integer) 2
127.0.0.1:6379> JSON.STRAPPEND k1 .a.* '"a"'
(integer) 3
127.0.0.1:6379> JSON.STRAPPEND k1 .b.* '"a"'
(integer) 2
127.0.0.1:6379> JSON.STRAPPEND k1 .c.* '"a"'
(integer) 3
127.0.0.1:6379> JSON.STRAPPEND k1 .c.b '"a"'
(integer) 4
127.0.0.1:6379> JSON.STRAPPEND k1 .d.* '"a"'
(integer) 2
```

## JSON.STRLEN

Gets the lengths of the JSON string values at the path.

Syntax

```
JSON.STRLEN <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

### Return

If the path is enhanced syntax:

- Array of integers that represents the length of the string value at each path.
- If a value is not a string, its corresponding return value is null.
- Null if the document key does not exist.

If the path is restricted syntax:

- Integer, the string's length.
- If multiple string values are selected, the command returns the first string's length.
- `WRONGTYPE` error if the value at the path is not a string.

- NONEXISTENT error if the path does not exist.
- Null if the document key does not exist.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 $ '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"bb"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.STRLEN k1 $.a.a
1) (integer) 1
127.0.0.1:6379> JSON.STRLEN k1 $.a.*
1) (integer) 1
127.0.0.1:6379> JSON.STRLEN k1 $.c.*
1) (integer) 1
2) (integer) 2
127.0.0.1:6379> JSON.STRLEN k1 $.c.b
1) (integer) 2
127.0.0.1:6379> JSON.STRLEN k1 $.d.*
1) (nil)
2) (integer) 1
3) (nil)
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 $ '{"a":{"a":"a"}, "b":{"a":"a", "b":1}, "c":{"a":"a",
 "b":"bb"}, "d":{"a":1, "b":"b", "c":3}}'
OK
127.0.0.1:6379> JSON.STRLEN k1 .a.a
(integer) 1
127.0.0.1:6379> JSON.STRLEN k1 .a.*
(integer) 1
127.0.0.1:6379> JSON.STRLEN k1 .c.*
(integer) 1
127.0.0.1:6379> JSON.STRLEN k1 .c.b
(integer) 2
127.0.0.1:6379> JSON.STRLEN k1 .d.*
(integer) 1
```

## JSON.TOGGLE

Toggles Boolean values between true and false at the path.

Syntax

```
JSON.TOGGLE <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

### Return

If the path is enhanced syntax:

- Array of integers (0 - false, 1 - true) that represent the resulting Boolean value at each path.
- If a value is a not a Boolean value, its corresponding return value is null.
- NONEXISTENT if the document key does not exist.

If the path is restricted syntax:

- String ("true"/"false") that represents the resulting Boolean value.
- NONEXISTENT if the document key does not exist.
- WRONGTYPE error if the value at the path is not a Boolean value.

### Examples

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '{"a":true, "b":false, "c":1, "d":null, "e":"foo", "f":
[], "g":{}}'
OK
127.0.0.1:6379> JSON.TOGGLE k1 $.*
1) (integer) 0
2) (integer) 1
3) (nil)
4) (nil)
5) (nil)
```

```
6) (nil)
7) (nil)
127.0.0.1:6379> JSON.TOGGLE k1 $.*
1) (integer) 1
2) (integer) 0
3) (nil)
4) (nil)
5) (nil)
6) (nil)
7) (nil)
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 . true
OK
127.0.0.1:6379> JSON.TOGGLE k1
"false"
127.0.0.1:6379> JSON.TOGGLE k1
"true"

127.0.0.1:6379> JSON.SET k2 . '{"isAvailable": false}'
OK
127.0.0.1:6379> JSON.TOGGLE k2 .isAvailable
"true"
127.0.0.1:6379> JSON.TOGGLE k2 .isAvailable
"false"
```

## JSON.TYPE

Reports the type of values at the given path.

Syntax

```
JSON.TYPE <key> [path]
```

- key (required) – A Valkey or Redis OSS key of JSON document type.
- path (optional) – A JSON path. Defaults to the root if not provided.

**Return**

If the path is enhanced syntax:

- Array of strings that represent the type of value at each path. The type is one of {"null", "boolean", "string", "number", "integer", "object" and "array"}.
- If a path does not exist, its corresponding return value is null.
- Empty array if the document key does not exist.

If the path is restricted syntax:

- String, type of the value
- Null if the document key does not exist.
- Null if the JSON path is invalid or does not exist.

**Examples**

Enhanced path syntax:

```
127.0.0.1:6379> JSON.SET k1 . '[1, 2.3, "foo", true, null, {}, []]'
OK
127.0.0.1:6379> JSON.TYPE k1 $[*]
1) integer
2) number
3) string
4) boolean
5) null
6) object
7) array
```

Restricted path syntax:

```
127.0.0.1:6379> JSON.SET k1 .
 '{"firstName":"John","lastName":"Smith","age":27,"weight":135.25,"isAlive":true,"address":
{"street":"21 2nd Street","city":"New
 York","state":"NY","zipcode":"10021-3100"},"phoneNumbers":
[{"type":"home","number":"212 555-1234"},{"type":"office","number":"646
 555-4567"}],"children":[],"spouse":null}'
OK
127.0.0.1:6379> JSON.TYPE k1
```

```
object
127.0.0.1:6379> JSON.TYPE k1 .children
array
127.0.0.1:6379> JSON.TYPE k1 .firstName
string
127.0.0.1:6379> JSON.TYPE k1 .age
integer
127.0.0.1:6379> JSON.TYPE k1 .weight
number
127.0.0.1:6379> JSON.TYPE k1 .isAlive
boolean
127.0.0.1:6379> JSON.TYPE k1 .spouse
null
```

# Tagging your ElastiCache resources

To help you manage your clusters and other ElastiCache resources, you can assign your own metadata to each resource in the form of tags. Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or environment. This is useful when you have many resources of the same type—you can quickly identify a specific resource based on the tags that you've assigned to it. This topic describes tags and shows you how to create them.

> **Warning**
>
> As a best practice, we recommend that you do not include sensitive data in your tags.

## Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize your AWS resources in different ways, for example, by purpose or owner. For example, you could define a set of tags for your account's ElastiCache clusters that helps you track each instance's owner and user group.

We recommend that you devise a set of tag keys that meets your needs for each resource type. Using a consistent set of tag keys makes it easier for you to manage your resources. You can search and filter the resources based on the tags you add. For more information about how to implement an effective resource tagging strategy, see the AWS whitepaper Tagging Best Practices.

Tags don't have any semantic meaning to ElastiCache and are interpreted strictly as a string of characters. Also, tags are not automatically assigned to your resources. You can edit tag keys and values, and you can remove tags from a resource at any time. You can set the value of a tag to `null`. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value. If you delete a resource, any tags for the resource are also deleted. Furthermore, if you add or delete tags on a replication group, all nodes in that replication group will also have their tags added or removed.

You can work with tags using the AWS Management Console, the AWS CLI, and the ElastiCache API.

If you're using IAM, you can control which users in your AWS account have permission to create, edit, or delete tags. For more information, see Resource-level permissions.

## Resources you can tag

You can tag most ElastiCache resources that already exist in your account. The table below lists the resources that support tagging. If you're using the AWS Management Console, you can apply tags to resources by using the Tag Editor. Some resource screens enable you to specify tags for a resource when you create the resource; for example, a tag with a key of Name and a value that you specify. In most cases, the console applies the tags immediately after the resource is created (rather than during resource creation). The console may organize resources according to the **Name** tag, but this tag doesn't have any semantic meaning to the ElastiCache service.

Additionally, some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, we roll back the resource creation process. This ensures that resources are either created with tags or not created at all, and that no resources are left untagged at any time. By tagging resources at the time of creation, you can eliminate the need to run custom tagging scripts after resource creation.

If you're using the Amazon ElastiCache API, the AWS CLI, or an AWS SDK, you can use the `Tags` parameter on the relevant ElastiCache API action to apply tags. They are:

- `CreateServerlessCache`
- `CreateCacheCluster`
- `CreateReplicationGroup`
- `CopyServerlessCacheSnapshot`
- `CopySnapshot`
- `CreateCacheParameterGroup`

- `CreateCacheSecurityGroup`
- `CreateCacheSubnetGroup`
- `CreateServerlessCacheSnapshot`
- `CreateSnapshot`
- `CreateUserGroup`
- `CreateUser`
- `PurchaseReservedCacheNodesOffering`

The following table describes the ElastiCache resources that can be tagged, and the resources that can be tagged on creation using the ElastiCache API, the AWS CLI, or an AWS SDK.

**Tagging support for ElastiCache resources**

| Supports tags | Supports tagging on creation |
|---|---|
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |

| Supports tags | Supports tagging on creation |
|---|---|
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |
| Yes | Yes |

> **Note**
>
> You cannot tag Global Datastores. However, you can tag the individual member replication groups and clusters within a Global Datastore.

You can apply tag-based resource-level permissions in your IAM policies to the ElastiCache API actions that support tagging on creation to implement granular control over the users and groups that can tag resources on creation. Your resources are properly secured from creation—tags that are applied immediately to your resources. Therefore any tag-based resource-level permissions controlling the use of resources are immediately effective. Your resources can be tracked and reported on more accurately. You can enforce the use of tagging on new resources, and control which tag keys and values are set on your resources.

For more information, see Tagging resources examples.

For more information about tagging your resources for billing, see Monitoring costs with cost allocation tags.

## Tagging caches and snapshots

The following rules apply to tagging as part of request operations:

- **CreateReplicationGroup**:
  - If the `--primary-cluster-id` and `--tags` parameters are included in the request, the request tags will be added to the replication group and propagate to all clusters in the

replication group. If the primary cluster has existing tags, these will be overwritten with the request tags to have consistent tags across all nodes.

If there are no request tags, the primary cluster tags will be added to the replication group and propagated to all clusters.

- If the `--snapshot-name` or `--serverless-cache-snapshot-name` is supplied:

  If tags are included in the request, the replication group will be tagged only with those tags. If no tags are included in the request, the snapshot tags will be added to the replication group.

- If the `--global-replication-group-id` is supplied:

  If tags are included in the request, the request tags will be added to the replication group and propagate to all clusters.

- **CreateCacheCluster** :
  - If the `--replication-group-id` is supplied:

    If tags are included in the request, the cluster will be tagged only with those tags. If no tags are included in the request, the cluster will inherit the replication group tags instead of the primary cluster's tags.

  - If the `--snapshot-name` is supplied:

    If tags are included in the request, the cluster will be tagged only with those tags. If no tags are included in the request, the snapshot tags will be added to the cluster.

- **CreateServerlessCache** :
  - If tags are included in the request, only the request tags will be added to the serverless cache.
- **CreateSnapshot** :
  - If the `--replication-group-id` is supplied:

    If tags are included in the request, only the request tags will be added to the snapshot. If no tags are included in the request, the replication group tags will be added to the snapshot.

  - If the `--cache-cluster-id` is supplied:

    If tags are included in the request, only the request tags will be added to the snapshot. If no tags are included in the request, the cluster tags will be added to the snapshot.

  - For automatic snapshots:

    Tags will propagate from the replication group tags.

- **CreateServerlessCacheSnapshot** :
  - If tags are included in the request, only the request tags will be added to the serverless cache snapshot.
- **CopySnapshot** :
  - If tags are included in the request, only the request tags will be added to the snapshot. If no tags are included in the request, the source snapshot tags will be added to the copied snapshot.
- **CopyServerlessCacheSnapshot** :
  - If tags are included in the request, only the request tags will be added to the serverless cache snapshot.
- **AddTagsToResource** and **RemoveTagsFromResource** :
  - Tags will be added/removed from the replication group and the action will be propagated to all clusters in the replication group.

> **Note**
>
> **AddTagsToResource** and **RemoveTagsFromResource** cannot be used for default parameter and security groups.

- **IncreaseReplicaCount** and **ModifyReplicationGroupShardConfiguration**:
  - All new clusters added to the replication group will have the same tags applied as the replication group.

## Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8.
- Maximum value length – 256 Unicode characters in UTF-8.
- Although ElastiCache allows for any character in its tags, other services can be restrictive. The allowed characters across services are: letters, numbers, and spaces representable in UTF-8, and the following characters: + - = . _ : / @
- Tag keys and values are case-sensitive.

- The `aws:` prefix is reserved for AWS use. If a tag has a tag key with this prefix, then you can't edit or delete the tag's key or value. Tags with the `aws:` prefix do not count against your tags per resource limit.

You can't terminate, stop, or delete a resource based solely on its tags; you must specify the resource identifier. For example, to delete snapshots that you tagged with a tag key called `DeleteMe`, you must use the `DeleteSnapshot` action with the resource identifiers of the snapshots, such as `snap-1234567890abcdef0`.

For more information on ElastiCache resources you can tag, see Resources you can tag.

## Tagging resources examples

- Creating a serverless cache using tags. This example uses Memcached as the engine.

```
aws elasticache create-serverless-cache \
    --serverless-cache-name CacheName \
    --engine memcached
    --tags Key="Cost Center", Value="1110001" Key="project",Value="XYZ"
```

- Adding tags to a serverless cache

```
aws elasticache add-tags-to-resource \
--resource-name arn:aws:elasticache:us-east-1:111111222233:serverlesscache:my-cache \
--tags Key="project",Value="XYZ" Key="Elasticache",Value="Service"
```

- Adding tags to a Replication Group.

```
aws elasticache add-tags-to-resource \
--resource-name arn:aws:elasticache:us-east-1:111111222233:replicationgroup:my-rg \
--tags Key="project",Value="XYZ" Key="Elasticache",Value="Service"
```

- Creating a Cache Cluster using tags.

```
aws elasticache create-cache-cluster \
--cluster-id testing-tags \
--cluster-description cluster-test \
--cache-subnet-group-name test \
--cache-node-type cache.t2.micro \
--engine valkey \
--tags Key="project",Value="XYZ" Key="Elasticache",Value="Service"
```

- Creating a Cache Cluster using tags. This example uses Redis as the engine.

```
aws elasticache create-cache-cluster \
--cluster-id testing-tags \
--cluster-description cluster-test \
--cache-subnet-group-name test \
--cache-node-type cache.t2.micro \
--engine valkey \
--tags Key="project",Value="XYZ" Key="Elasticache",Value="Service"
```

- Creating a serverless snapshot with tags. This example uses Memcached as the engine.

```
aws elasticache create-serverless-cache-snapshot \
--serverless-cache-name testing-tags \
--serverless-cache-snapshot-name bkp-testing-tags-scs \
--tags Key="work",Value="foo"
```

- Creating a Snapshot with tags.

  Snapshots are currently only available for Redis. For this case, if you add tags on request, even if the replication group contains tags, the snapshot will receive only the request tags.

```
aws elasticache create-snapshot \
--replication-group-id testing-tags \
--snapshot-name bkp-testing-tags-rg \
--tags Key="work",Value="foo"
```

## Tag-Based access control policy examples

1. Allowing `AddTagsToResource` action to a cluster only if the cluster has the tag Project=XYZ.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticache:AddTagsToResource",
```

```
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Project": "XYZ"
                }
            }
        }
    ]
}
```

2. Allowing to `RemoveTagsFromResource` action from a replication group if it contains the tags Project and Service and keys are different from Project and Service.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticache:RemoveTagsFromResource",
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Service": "Elasticache",
                    "aws:ResourceTag/Project": "XYZ"
                },
                "ForAnyValue:StringNotEqualsIgnoreCase": {
                    "aws:TagKeys": [
                        "Project",
                        "Service"
                    ]
                }
            }
        }
    ]
}
```

3. Allowing `AddTagsToResource` to any resource only if tags are different from Project and Service.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticache:AddTagsToResource",
            "Resource": [
                "arn:aws:elasticache:*:*:*:*"
            ],
            "Condition": {
                "ForAnyValue:StringNotEqualsIgnoreCase": {
                    "aws:TagKeys": [
                        "Service",
                        "Project"
                    ]
                }
            }
        }
    ]
}
```

4. Denying `CreateReplicationGroup` action if request has `Tag Project=Foo`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "elasticache:CreateReplicationGroup",
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Project": "Foo"
                }
            }
        }
    ]
```

```
}
```

5. Denying CopySnapshot action if source snapshot has tag Project=XYZ and request tag is Service=Elasticache.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "elasticache:CopySnapshot",
            "Resource": [
                "arn:aws:elasticache:*:*:snapshot:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Project": "XYZ",
                    "aws:RequestTag/Service": "Elasticache"
                }
            }
        }
    ]
}
```

6. Denying CreateCacheCluster action if the request tag Project is missing or is not equal to Dev, QA or Prod.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
          {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*",
                "arn:aws:elasticache:*:*:securitygroup:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
```

```
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "Null": {
                    "aws:RequestTag/Project": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:AddTagsToResource"
            ],
            "Resource": "arn:aws:elasticache:*:*:cluster:*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Project": [
                        "Dev",
                        "Prod",
                        "QA"
                    ]
                }
            }
        }
    ]
}
```

For related information on condition keys, see Using condition keys.

## Monitoring costs with cost allocation tags

When you add cost allocation tags to your resources in Amazon ElastiCache, you can track costs by grouping expenses on your invoices by resource tag values.

An ElastiCache cost allocation tag is a key-value pair that you define and associate with an ElastiCache resource. The key and value are case-sensitive. You can use a tag key to define a category, and the tag value can be an item in that category. For example, you might define a tag key of `CostCenter` and a tag value of `10010`, indicating that the resource is assigned to the 10010 cost center. You can also use tags to designate resources as being used for test or production by using a key such as `Environment` and values such as `test` or `production`. We recommend that you use a consistent set of tag keys to make it easier to track costs associated with your resources.

Use cost allocation tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services.

You can also combine tags to track costs at a greater level of detail. For example, to track your service costs by region you might use the tag keys `Service` and `Region`. On one resource you might have the values `ElastiCache` and `Asia Pacific (Singapore)`, and on another resource the values `ElastiCache` and `Europe (Frankfurt)`. You can then see your total ElastiCache costs broken out by region. For more information, see Use Cost Allocation Tags in the *AWS Billing User Guide*.

You can add ElastiCache cost allocation tags to ElastiCache node-based clusters. When you add, list, modify, copy, or remove a tag, the operation is applied only to the specified cluster.

**Characteristics of ElastiCache cost allocation tags**

- Cost allocation tags are applied to ElastiCache resources which are specified in CLI and API operations as an ARN. The resource-type will be a "cluster".

  Sample ARN: `arn:aws:elasticache:<region>:<customer-id>:<resource-type>:<resource-name>`

  Sample arn: `arn:aws:elasticache:us-west-2:1234567890:cluster:my-cluster`
- The tag key is the required name of the tag. The key's string value can be from 1 to 128 Unicode characters long and cannot be prefixed with `aws:`. The string can contain only the set of Unicode letters, digits, blank spaces, underscores ( _ ), periods ( . ), colons ( : ), backslashes ( \ ), equal signs ( = ), plus signs ( + ), hyphens ( - ), or at signs ( @ ).

- The tag value is the optional value of the tag. The value's string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with `aws:`. The string can contain only the set of Unicode letters, digits, blank spaces, underscores ( _ ), periods ( . ), colons ( : ), backslashes ( \ ), equal signs ( = ), plus signs ( + ), hyphens ( - ), or at signs ( @ ).

- An ElastiCache resource can have a maximum of 50 tags.

- Values do not have to be unique in a tag set. For example, you can have a tag set where the keys `Service` and `Application` both have the value `ElastiCache`.

AWS does not apply any semantic meaning to your tags. Tags are interpreted strictly as character strings. AWS does not automatically set any tags on any ElastiCache resource.

## Managing your cost allocation tags using the AWS CLI

You can use the AWS CLI to add, modify, or remove cost allocation tags.

Cost allocation tags are applied to ElastiCache clusters. The cluster to be tagged is specified using an ARN (Amazon Resource Name).

Sample arn: `arn:aws:elasticache:us-west-2:1234567890:cluster:my-cluster`

**Topics**


- Listing tags using the AWS CLI
- Adding tags using the AWS CLI
- Modifying tags using the AWS CLI
- Removing tags using the AWS CLI


### Listing tags using the AWS CLI

You can use the AWS CLI to list tags on an existing ElastiCache resource by using the list-tags-for-resource operation.

The following code uses the AWS CLI to list the tags on the Memcached cluster `my-cluster` in the us-west-2 region.

For Linux, macOS, or Unix:

```
aws elasticache list-tags-for-resource \
  --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster
```

For Windows:

```
aws elasticache list-tags-for-resource ^
  --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster
```

The following code uses the AWS CLI to list the tags on the Valkey or Redis OSS node `my-cluster-001` in the `my-cluster` cluster in region us-west-2.

For Linux, macOS, or Unix:

```
aws elasticache list-tags-for-resource \
  --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001
```

For Windows:

```
aws elasticache list-tags-for-resource ^
  --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001
```

Output from this operation will look something like the following, a list of all the tags on the resource.

```
{
   "TagList": [
      {
         "Value": "10110",
         "Key": "CostCenter"
      },
      {
         "Value": "EC2",
         "Key": "Service"
      }
   ]
}
```

If there are no tags on the resource, the output will be an empty TagList.

```
{
   "TagList": []
}
```

For more information, see the AWS CLI for ElastiCache list-tags-for-resource.

## Adding tags using the AWS CLI

You can use the AWS CLI to add tags to an existing ElastiCache resource by using the add-tags-to-resource CLI operation. If the tag key does not exist on the resource, the key and value are added to the resource. If the key already exists on the resource, the value associated with that key is updated to the new value.

The following code uses the AWS CLI to add the keys `Service` and `Region` with the values `elasticache` and `us-west-2` respectively to the node `my-cluster-001` in the cluster `my-cluster` in region us-west-2.

### Memcached

For Linux, macOS, or Unix:

```
aws elasticache add-tags-to-resource \
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster \
 --tags Key=Service,Value=elasticache \
        Key=Region,Value=us-west-2
```

For Windows:

```
aws elasticache add-tags-to-resource ^
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster ^
 --tags Key=Service,Value=elasticache ^
        Key=Region,Value=us-west-2
```

### Redis

For Linux, macOS, or Unix:

```
aws elasticache add-tags-to-resource \
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001 \
 --tags Key=Service,Value=elasticache \
```

```
        Key=Region,Value=us-west-2
```

For Windows:

```
aws elasticache add-tags-to-resource ^
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001 ^
 --tags Key=Service,Value=elasticache ^
        Key=Region,Value=us-west-2
```

Output from this operation will look something like the following, a list of all the tags on the resource following the operation.

```
{
   "TagList": [
      {
         "Value": "elasticache",
         "Key": "Service"
      },
      {
         "Value": "us-west-2",
         "Key": "Region"
      }
   ]
}
```

For more information, see the AWS CLI for ElastiCache add-tags-to-resource.

You can also use the AWS CLI to add tags to a cluster when you create a new cluster by using the operation create-cache-cluster. You cannot add tags when creating a cluster using the ElastiCache management console. After the cluster is created, you can then use the console to add tags to the cluster.

## Modifying tags using the AWS CLI

You can use the AWS CLI to modify the tags on an ElastiCache cluster.

To modify tags:

- Use add-tags-to-resource to either add a new tag and value or to change the value associated with an existing tag.
- Use remove-tags-from-resource to remove specified tags from the resource.

Output from either operation will be a list of tags and their values on the specified cluster.

### Removing tags using the AWS CLI

You can use the AWS CLI to remove tags from an existing ElastiCache for Memcached cluster by using the remove-tags-from-resource operation.

For Memcached, the following code uses the AWS CLI to remove the tags with the keys `Service` and `Region` from the node `my-cluster-001` in the cluster `my-cluster` in the us-west-2 region.

For Linux, macOS, or Unix:

```
aws elasticache remove-tags-from-resource \
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster \
 --tag-keys PM Service
```

For Windows:

```
aws elasticache remove-tags-from-resource ^
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster ^
 --tag-keys PM Service
```

For Redis OSS, the following code uses the AWS CLI to remove the tags with the keys `Service` and `Region` from the node `my-cluster-001` in the cluster `my-cluster` in the us-west-2 region.

For Linux, macOS, or Unix:

```
aws elasticache remove-tags-from-resource \
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001 \
 --tag-keys PM Service
```

For Windows:

```
aws elasticache remove-tags-from-resource ^
 --resource-name arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001 ^
 --tag-keys PM Service
```

Output from this operation will look something like the following, a list of all the tags on the resource following the operation.

```
{
```

```
   "TagList": []
}
```

For more information, see the AWS CLI for ElastiCache remove-tags-from-resource.

## Managing your cost allocation tags using the ElastiCache API

You can use the ElastiCache API to add, modify, or remove cost allocation tags.

Cost allocation tags are applied to ElastiCache for Memcached clusters. The cluster to be tagged is specified using an ARN (Amazon Resource Name).

Sample arn: `arn:aws:elasticache:us-west-2:1234567890:cluster:my-cluster`

**Topics**


- Listing tags using the ElastiCache API
- Adding tags using the ElastiCache API
- Modifying tags using the ElastiCache API
- Removing tags using the ElastiCache API


### Listing tags using the ElastiCache API

You can use the ElastiCache API to list tags on an existing resource by using the ListTagsForResource operation.

For Memcached, the following code uses the ElastiCache API to list the tags on the resource `my-cluster` in the us-west-2 region.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=ListTagsForResource
   &ResourceName=arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Version=2015-02-02
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

For Redis OSS, the following code uses the ElastiCache API to list the tags on the resource `my-cluster-001` in the us-west-2 region.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=ListTagsForResource
   &ResourceName=arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Version=2015-02-02
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

## Adding tags using the ElastiCache API

You can use the ElastiCache API to add tags to an existing ElastiCache cluster by using the AddTagsToResource operation. If the tag key does not exist on the resource, the key and value are added to the resource. If the key already exists on the resource, the value associated with that key is updated to the new value.

The following code uses the ElastiCache API to add the keys `Service` and `Region` with the values `elasticache` and `us-west-2` respectively. For Memcached, this is applied to the resource `my-cluster`. For Redis OSS, this is applied to the resource `my-cluster-001` in the us-west-2 region.

### Memcached

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=AddTagsToResource
   &ResourceName=arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Tags.member.1.Key=Service
   &Tags.member.1.Value=elasticache
   &Tags.member.2.Key=Region
   &Tags.member.2.Value=us-west-2
   &Version=2015-02-02
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

### Redis

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=AddTagsToResource
   &ResourceName=arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001
   &SignatureVersion=4
```

```
   &SignatureMethod=HmacSHA256
   &Tags.member.1.Key=Service
   &Tags.member.1.Value=elasticache
   &Tags.member.2.Key=Region
   &Tags.member.2.Value=us-west-2
   &Version=2015-02-02
   &Timestamp=20150202T192317Z
   &X-Amz-Credential=<credential>
```

For more information, see AddTagsToResource in the *Amazon ElastiCache API Reference*.

## Modifying tags using the ElastiCache API

You can use the ElastiCache API to modify the tags on an ElastiCache cluster.

To modify the value of a tag:

- Use AddTagsToResource operation to either add a new tag and value or to change the value of an existing tag.
- Use RemoveTagsFromResource to remove tags from the resource.

Output from either operation will be a list of tags and their values on the specified resource.

Use RemoveTagsFromResource to remove tags from the resource.

## Removing tags using the ElastiCache API

You can use the ElastiCache API to remove tags from an existing ElastiCache for Memcached cluster by using the RemoveTagsFromResource operation.

The following code uses the ElastiCache API to remove the tags with the keys `Service` and `Region` from the node `my-cluster-001` in the cluster `my-cluster` in region us-west-2.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=RemoveTagsFromResource
   &ResourceName=arn:aws:elasticache:us-west-2:0123456789:cluster:my-cluster-001
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &TagKeys.member.1=Service
   &TagKeys.member.2=Region
   &Version=2015-02-02
```

```
    &Timestamp=20150202T192317Z
    &X-Amz-Credential=<credential>
```

# Using the Amazon ElastiCache Well-Architected Lens

This section describes the Amazon ElastiCache Well-Architected Lens, a collection of design principles and guidance for designing well-architected ElastiCache workloads.

- The ElastiCache Lens is additive to the AWS Well-Architected Framework.
- Each Pillar has a set of questions to help start the discussion around an ElastiCache Architecture.
  - Each question has a number of leading practices along with their scores for reporting.
    - *Required* - Necessary before going to prod (absent being a high risk)
    - *Best* - Best possible state a customer could be
    - *Good* - What we recommend customers to have (absent being a medium risk)
- Well-Architected terminology
  - Component – Code, configuration and AWS Resources that together deliver against a requirement. Components interact with other components, and often equate to a service in microservice architectures.
  - Workload - A set of components that together deliver business value. Examples of workloads are marketing websites, e-commerce websites, the back-ends for a mobile app, analytic platforms, etc.

> **Note**
>
> This guide has not been updated to include information on ElastiCache serverless caching and the new Valkey engine.

### Topics


- Amazon ElastiCache Well-Architected Lens Operational Excellence Pillar
- Amazon ElastiCache Well-Architected Lens Security Pillar
- Amazon ElastiCache Well-Architected Lens Reliability Pillar
- Amazon ElastiCache Well-Architected Lens Performance Efficiency Pillar
- Amazon ElastiCache Well-Architected Lens Cost Optimization Pillar

# Amazon ElastiCache Well-Architected Lens Operational Excellence Pillar

The operational excellence pillar focuses on running and monitoring systems to deliver business value, and continually improving processes and procedures. Key topics include automating changes, responding to events, and defining standards to manage daily operations.

**Topics**


- OE 1: How do you understand and respond to alerts and events triggered by your ElastiCache cluster?
- OE 2: When and how do you scale your existing ElastiCache clusters?
- OE 3: How do you manage your ElastiCache cluster resources and maintain your cluster up-to-date?
- OE 4: How do you manage clients' connections to your ElastiCache clusters?
- OE 5: How do you deploy ElastiCache Components for a Workload?
- OE 6: How do you plan for and mitigate failures?
- OE 7: How do you troubleshoot Valkey or Redis OSS engine events?


## OE 1: How do you understand and respond to alerts and events triggered by your ElastiCache cluster?

**Question-level introduction:** When you operate ElastiCache clusters you can optionally receive notifications and alerts when specific events occur. ElastiCache, by default, logs events that relate to your resources, such as a failover, node replacement, scaling operation, scheduled maintenance, and more. Each event includes the date and time, the source name and source type, and a description.

**Question-level benefit:** Being able to understand and manage the underlying reasons behind the events that trigger alerts generated by your cluster enables you to operate more effectively and respond to events appropriately.

- **[Required]** Review the events generated by ElastiCache on the ElastiCache console (after selecting your region) or using the Amazon Command Line Interface (AWS CLI) describe-events command and the ElastiCache API. Configure ElastiCache to send notifications for important cluster events using Amazon Simple Notification Service (Amazon SNS). Using Amazon SNS with your clusters allows you to programmatically take actions upon ElastiCache events.

- There are two large categories of events: current and scheduled events. The list of current events includes: resource creation and deletion, scaling operations, failover, node reboot, snapshot created, cluster's parameter modification, CA certificate renewal, failure events (cluster provisioning failure - VPC or ENI-, scaling failures - ENI-, and snapshot failures). The list of scheduled events includes: node scheduled for replacement during the maintenance window and node replacement rescheduled.
- Although you may not need to react immediately to some of these events, it is critical to first look at all failure events:
  - ElastiCache:AddCacheNodeFailed
  - ElastiCache:CacheClusterProvisioningFailed
  - ElastiCache:CacheClusterScalingFailed
  - ElastiCache:CacheNodesRebooted
  - ElastiCache:SnapshotFailed (Valkey or Redis OSS only)
- **[Resources]:**
  - Managing ElastiCache Amazon SNS notifications
  - Event Notifications and Amazon SNS
- **[Best]** To automate responses to events, leverage AWS products and services capabilities such as SNS and Lambda Functions. Follow best practices by making small, frequent, reversible changes, as code to evolve your operations over time. You should use Amazon CloudWatch metrics to monitor your clusters.

  **[Resources]:** Monitor ElastiCache (cluster mode disabled) read replica endpoints using AWS Lambda, Amazon Route 53, and Amazon SNS for a use case that uses Lambda and SNS.

## OE 2: When and how do you scale your existing ElastiCache clusters?

**Question-level introduction:** Right-sizing your ElastiCache cluster is a balancing act that needs to be evaluated every time there are changes to the underlying workload types. Your objective is to operate with the right sized environment for your workload.

**Question-level benefit:** Over-utilization of your resources may result in elevated latency and overall decreased performance. Under-utilization, on the other hand, may result in over-provisioned resources at non-optimal cost optimization. By right-sizing your environments you can strike a balance between performance efficiency and cost optimization. To remediate over or under utilization of your resources, ElastiCache can scale in two dimensions. You can scale vertically by

increasing or decreasing node capacity. You can also scale horizontally by adding and removing nodes.

- **[Required]** CPU and network over-utilization on primary nodes should be addressed by offloading and redirecting the read operations to replica nodes. Use replica nodes for read operations to reduce primary node utilization. This can be configured in your Valkey or Redis OSS client library by connecting to the ElastiCache reader endpoint for cluster mode disabled, or by using the READONLY command for cluster mode enabled.

  **[Resources]:**

  - Finding connection endpoints in ElastiCache
  - Cluster Right-Sizing
  - READONLY Command
- **[Required]** Monitor the utilization of critical cluster resources such as CPU, memory, and network. The utilization of these specific cluster resources needs to be tracked to inform your decision to scale, and the type of scaling operation. For ElastiCache cluster mode disabled, primary and replica nodes can scale vertically. Replica nodes can also scale horizontally from 0 to 5 nodes. For cluster mode enabled, the same applies within each shard of your cluster. In addition, you can increase or reduce the number of shards.

  **[Resources]:**

  - Monitoring best practices with ElastiCache using Amazon CloudWatch
  - Scaling ElastiCache Clusters for Valkey and Redis OSS
  - Scaling ElastiCache Clusters for Memcached
- **[Best]** Monitoring trends over time can help you detect workload changes that would remain unnoticed if monitored at a particular point in time. To detect longer term trends, use CloudWatch metrics to scan for longer time ranges. The learnings from observing extended periods of CloudWatch metrics should inform your forecast around cluster resources utilization. CloudWatch data points and metrics are available for up to 455 days.

  **[Resources]:**

  - Monitoring ElastiCache with CloudWatch Metrics
  - Monitoring Memcached with CloudWatch Metrics
  - Monitoring best practices with ElastiCache using Amazon CloudWatch

- **[Best]** If your ElastiCache resources are created with CloudFormation it is best practice to perform changes using CloudFormation templates to preserve operational consistency and avoid unmanaged configuration changes and stack drifts.

  **[Resources]:**

  - ElastiCache resource type reference for CloudFormation
- **[Best]** Automate your scaling operations using cluster operational data and define thresholds in CloudWatch to setup alarms. Use CloudWatch Events and Simple Notification Service (SNS) to trigger Lambda functions and execute an ElastiCache API to scale your clusters automatically. An example would be to add a shard to your cluster when the `EngineCPUUtilization` metric reaches 80% for an extended period of time. Another option would be to use `DatabaseMemoryUsedPercentages` for a memory-based threshold.

  **[Resources]:**

  - Using Amazon CloudWatch Alarms
  - What are Amazon CloudWatch events?
  - Using AWS Lambda with Amazon Simple Notification Service
  - ElastiCache API Reference

## OE 3: How do you manage your ElastiCache cluster resources and maintain your cluster up-to-date?

**Question-level introduction:** When operating at scale, it is essential that you are able to pinpoint and identify all your ElastiCache resources. When rolling out new application features you need to create cluster version symmetry across all your ElastiCache environment types: dev, testing, and production. Resource attributes allow you to separate environments for different operational objectives, such as when rolling out new features and enabling new security mechanisms.

**Question-level benefit:** Separating your development, testing, and production environments is best operational practice. It is also best practice that your clusters and nodes across environments have the latest software patches applied using well understood and documented processes. Taking advantage of native ElastiCache features enables your engineering team to focus on meeting business objectives and not on ElastiCache maintenance.

- **[Best]** Run on the latest engine version available and apply the Self-Service Updates as quickly as they become available. ElastiCache automatically updates its underlying infrastructure during your specified maintenance window of the cluster. However, the nodes running in your clusters

are updated via Self-Service Updates. These updates can be of two types: security patches or minor software updates. Ensure you understand the difference between types of patches and when they are applied.

**[Resources]:**

- Self-Service Updates in Amazon ElastiCache
- Amazon ElastiCache Managed Maintenance and Service Updates Help Page

- **[Best]** Organize your ElastiCache resources using tags. Use tags on replication groups and not on individual nodes. You can configure tags to be displayed when you query resources and you can use tags to perform searches and apply filters. You should use Resource Groups to easily create and maintain collections of resources that share common sets of tags.

  **[Resources]:**

  - Tagging Best Practices
  - ElastiCache resource type reference for CloudFormation
  - Parameter Groups

## OE 4: How do you manage clients' connections to your ElastiCache clusters?

**Question-level introduction:** When operating at scale you need to understand how your clients connect with the ElastiCache cluster to manage your application operational aspects (such as response times).

**Question-level benefit:** Choosing the most appropriate connection mechanism ensures that your application does not disconnect due to connectivity errors, such as time-outs.

- **[Required]** Separate read from write operations and connect to the replica nodes to execute read operations. However, be aware when you separate the writes from the reads you will lose the ability to read a key immediately after writing it due to the asynchronous nature of the Valkey and Redis OSS replication. The WAIT command can be leveraged to improve real world data safety and force replicas to acknowledge writes before responding to clients, at an overall performance cost. Using replica nodes for read operations can be configured in your ElastiCache client library using the ElastiCache reader endpoint for cluster mode disabled. For cluster mode enabled, use the READONLY command. For many of the ElastiCache client libraries, READONLY is implemented by default or via a configuration setting.

  **[Resources]:**

- Finding connection endpoints in ElastiCache
- READONLY

- **[Required]** Use connection pooling. Establishing a TCP connection has a cost in CPU time on both client and server sides and pooling allows you to reuse the TCP connection.

  To reduce connection overhead, you should use connection pooling. With a pool of connections your application can re-use and release connections 'at will', without the cost of establishing the connection. You can implement connection pooling via your ElastiCache client library (if supported), with a Framework available for your application environment, or build it from the ground up.
- **[Best]** Ensure that the socket timeout of the client is set to at least one second (vs. the typical "none" default in several clients).
  - Setting the timeout value too low can lead to possible timeouts when the server load is high. Setting it too high can result in your application taking a long time to detect connection issues.
  - Control the volume of new connections by implementing connection pooling in your client application. This reduces latency and CPU utilization needed to open and close connections, and perform a TLS handshake if TLS is enabled on the cluster.

  **[Resources]:** Configure ElastiCache for higher availability
- **[Good]** Using pipelining (when your use cases allow it) can significantly boost the performance.
  - With pipelining you reduce the Round-Trip Time (RTT) between your application clients and the cluster and new requests can be processed even if the client has not yet read the previous responses.
  - With pipelining you can send multiple commands to the server without waiting for replies/ack. The downside of pipelining is that when you eventually fetch all the responses in bulk there may have been an error that you will not catch until the end.
  - Implement methods to retry requests when an error is returned that omits the bad request.

  **[Resources]:** Pipelining

## OE 5: How do you deploy ElastiCache Components for a Workload?

**Question-level introduction:** ElastiCache environments can be deployed manually through the AWS Console, or programmatically through APIs, CLI, toolkits, etc. Operational Excellence

best practices suggest automating deployments through code whenever possible. Additionally, ElastiCache clusters can either be isolated by workload or combined for cost optimization purposes.

**Question-level benefit:** Choosing the most appropriate deployment mechanism for your ElastiCache environments can improve Operation Excellence over time. It is recommended to perform operations as code whenever possible to minimize human error and increase repeatability, flexibility, and response time to events.

By understanding the workload isolation requirements, you can choose to have dedicated ElastiCache environments per workload or combine multiple workloads into single clusters, or combinations thereof. Understanding the tradeoffs can help strike a balance between Operational Excellence and Cost Optimization

- **[Required]** Understand the deployment options available to ElastiCache, and automate these procedures whenever possible. Possible avenues of automation include CloudFormation, AWS CLI/SDK, and APIs.

  **[Resources]:**

  - Amazon ElastiCache resource type reference
  - elasticache
  - Amazon ElastiCache API Reference
- **[Required]** For all workloads determine the level of cluster isolation needed.
  - **[Best]:** High Isolation – a 1:1 workload to cluster mapping. Allows for finest grained control over access, sizing, scaling, and management of ElastiCache resources on a per workload basis.
  - **[Better]:** Medium Isolation – M:1 isolated by purpose but perhaps shared across multiple workloads (for example a cluster dedicated to caching workloads, and another dedicated for messaging).
  - **[Good]:** Low Isolation – M:1 all purpose, fully shared. Recommended for workloads where shared access is acceptable.

## OE 6: How do you plan for and mitigate failures?

**Question-level introduction:** Operational Excellence includes anticipating failures by performing regular "pre-mortem" exercises to identify potential sources of failure so they can be removed or mitigated. ElastiCache offers a Failover API that allows for simulated node failure events, for testing purposes.

**Question-level benefit:** By testing failure scenarios ahead of time you can learn how they impact your workload. This allows for safe testing of response procedures and their effectiveness, as well as gets your team familiar with their execution.

**[Required]** Regularly perform failover testing in dev/test accounts. TestFailover

## OE 7: How do you troubleshoot Valkey or Redis OSS engine events?

**Question-level introduction:** Operational Excellence requires the ability to investigate both service-level and engine-level information to analyze the health and status of your clusters. ElastiCache can emit Valkey or Redis OSS engine logs to both Amazon CloudWatch and Amazon Kinesis Data Firehose.

**Question-level benefit:** Enabling Valkey or Redis OSS engine logs on ElastiCache clusters provides insight into events that impact the health and performance of clusters. Valkey or Redis OSS engine logs provide data directly from the engine that is not available through the ElastiCache events mechanism. Through careful observation of both ElastiCache events (see preceding OE-1) and engine logs, it is possible to determine an order of events when troubleshooting from both the ElastiCache service perspective and engine perspective.

- **[Required]** Ensure that Redis OSS engine logging functionality is enabled, which is available as of ElastiCache version 6.2 for Redis OSS and newer. This can be performed during cluster creation or by modifying the cluster after creation.
  - Determine whether Amazon CloudWatch Logs or Amazon Kinesis Data Firehose is the appropriate target for Redis OSS engine logs.
  - Select an appropriate target log within either CloudWatch or Kinesis Data Firehose to persist the logs. If you have multiple clusters, consider a different target log for each cluster as this will help isolate data when troubleshooting.

  **[Resources]:**

  - Log delivery: Log delivery
  - Logging destinations: Amazon CloudWatch Logs
  - Amazon CloudWatch Logs introduction: What is Amazon CloudWatch Logs?
  - Amazon Kinesis Data Firehose introduction: What Is Amazon Kinesis Data Firehose?
- **[Best]** If using Amazon CloudWatch Logs, consider leveraging Amazon CloudWatch Logs Insights to query Valkey or Redis OSS engine log for important information.

As an example, create a query against the CloudWatch Log group that contains the Valkey or Redis OSS engine logs that will return events with a LogLevel of 'WARNING', such as:

```
fields @timestamp, LogLevel, Message
| sort @timestamp desc
| filter LogLevel = "WARNING"
```

**[Resources]:**Analyzing log data with CloudWatch Logs Insights

## Amazon ElastiCache Well-Architected Lens Security Pillar

The security pillar focuses on protecting information and systems. Key topics include confidentiality and integrity of data, identifying and managing who can do what with privilege-based management, protecting systems, and establishing controls to detect security events.

**Topics**

- SEC 1: What steps are you taking in controlling authorized access to ElastiCache data?
- SEC 2: Do your applications require additional authorization to ElastiCache over and above networking-based controls?
- SEC 3: Is there a risk that commands can be executed inadvertently, causing data loss or failure?
- SEC 4: How do you ensure data encryption at rest with ElastiCache
- SEC 5: How do you encrypt in-transit data with ElastiCache?
- SEC 6: How do you restrict access to control plane resources?
- SEC 7: How do you detect and respond to security events?

### SEC 1: What steps are you taking in controlling authorized access to ElastiCache data?

**Question-level introduction:** All ElastiCache clusters are designed to be accessed from Amazon Elastic Compute Cloud instances in a VPC, serverless functions (AWS Lambda), or containers (Amazon Elastic Container Service). The most encountered scenario is to access an ElastiCache cluster from an Amazon Elastic Compute Cloud instance within the same Amazon Virtual Private Cloud (Amazon Virtual Private Cloud). Before you can connect to a cluster from an Amazon EC2 instance, you must authorize the Amazon EC2 instance to access the cluster. To access an ElastiCache cluster running in a VPC, it is necessary to grant network ingress to the cluster.

**Question-level benefit:** Network ingress into the cluster is controlled via VPC security groups. A security group acts as a virtual firewall for your Amazon EC2 instances to control incoming and outgoing traffic. Inbound rules control the incoming traffic to your instance, and outbound rules control the outgoing traffic from your instance. In the case of ElastiCache, when launching a cluster, it requires associating a security group. This ensures that inbound and outbound traffic rules are in place for all nodes that make up the cluster. Additionally, ElastiCache is configured to deploy on private subnets exclusively such that they are only accessible from via the VPC's private networking.

- **[Required]** The security group associated with your cluster controls network ingress and access to the cluster. By default, a security group will not have any inbound rules defined and, therefore, no ingress path to ElastiCache. To enable this, configure an inbound rule on the security group specifying source IP address/range, TCP type traffic and the port for your ElastiCache cluster (default port 6379 for ElastiCache for Valkey and Redis OSS for example). While it is possible to allow a very broad set of ingress sources, like all resources within a VPC (0.0.0.0/0), it is advised to be as granular as possible in defining the inbound rules such as authorizing only inbound access to Valkey or Redis OSS clients running on Amazon Amazon EC2 instances associated with a specific security group.

  **[Resources]:**

  - Subnets and subnet groups
  - Accessing your cluster or replication group
  - Control traffic to resources using security groups
  - Amazon Elastic Compute Cloud security groups for Linux instances
- **[Required]** AWS Identity and Access Management policies can be assigned to AWS Lambda functions allowing them to access ElastiCache data. To enable this feature, create an IAM execution role with the `AWSLambdaVPCAccessExecutionRole` permission, then assign the role to the AWS Lambda function.

  **[Resources]:** Configuring a Lambda function to access Amazon ElastiCache in an Amazon VPC: Tutorial: Configuring a Lambda function to access Amazon ElastiCache in an Amazon VPC

## SEC 2: Do your applications require additional authorization to ElastiCache over and above networking-based controls?

**Question-level introduction:** In scenarios where it is necessary to restrict or control access to clusters at an individual client level, it is recommended to authenticate via the AUTH command. ElastiCache authentication tokens, with optional user and user group management, enable ElastiCache to require a password before allowing clients to run commands and access keys, thereby improving data plane security.

**Question-level benefit:** To help keep your data secure, ElastiCache provides mechanisms to safeguard against unauthorized access of your data. This includes enforcing Role-Based Access Control (RBAC) AUTH, or AUTH token (password) be used by clients to connect to ElastiCache before performing authorized commands.

- **[Best]** For ElastiCache version 6.x and higher for Redis OSS, and ElastiCache version 7.2 and higher for Valkey, define authentication and authorization controls by defining user groups, users, and access strings. Assign users to user groups, then assign user groups to clusters. To utilize RBAC, it must be selected upon cluster creation, and in-transit encryption must be enabled. Ensure you are using a Valkey or Redis OSS client that supports TLS to be able to leverage RBAC.

  **[Resources]:**

  - Applying RBAC to a Replication Group for ElastiCache
  - Specifying Permissions Using an Access String
  - ACL
  - Supported ElastiCache versions
- **[Best]** For ElastiCache versions prior to 6.x for Redis OSS, in addition to setting strong token/password and maintaining a strict password policy for AUTH, it is best practice to rotate the password/token. ElastiCache can manage up to two (2) authentication tokens at any given time. You can also modify the cluster to explicitly require the use of authentication tokens.

  **[Resources]:** Modifying the AUTH token on an existing ElastiCache cluster

## SEC 3: Is there a risk that commands can be executed inadvertently, causing data loss or failure?

**Question-level introduction:** There are a number of Valkey or Redis OSS commands that can have adverse impacts on operations if executed by mistake or by malicious actors. These commands can have un-intended consequences from a performance and data safety perspective. For example a developer may routinely call the FLUSHALL command in a dev environment, and due to a mistake may inadvertently attempt to call this command on a production system, resulting in accidental data loss.

**Question-level benefit:** Beginning with ElastiCache version 5.0.3 for Redis OSS, you have the ability to rename certain commands that might be disruptive to your workload. Renaming the commands can help prevent them from being inadvertently executed on the cluster.

- **[Required]**

  **[Resources]:**

  - ElastiCache version 5.0.3 for Redis OSS (deprecated, use version 5.0.6)
  - ElastiCache version 5.0.3 for Redis OSS parameter changes
  - Redis OSS security

## SEC 4: How do you ensure data encryption at rest with ElastiCache

**Question-level introduction:** While ElastiCache is an in-memory data store, it is possible to encrypt any data that may be persisted (on storage) as part of standard operations of the cluster. This includes both scheduled and manual backups written to Amazon S3, as well as data saved to disk storage as a result of sync and swap operations. Instance types in the M6g and R6g families also feature always-on, in-memory encryption.

**Question-level benefit:** ElastiCache provides optional encryption at-rest to increase data security.

- **[Required]** At-rest encryption can be enabled on an ElastiCache cluster (replication group) only when it is created. An existing cluster cannot be modified to begin encrypting data at-rest. By default, ElastiCache will provide and manage the keys used in at-rest encryption.

  **[Resources]:**

  - At-Rest Encryption Constraints
  - Enabling At-Rest Encryption

- **[Best]** Leverage Amazon EC2 instance types that encrypt data while it is in memory (such as M6g or R6g). Where possible, consider managing your own keys for at-rest encryption. For more stringent data security environments, AWS Key Management Service (KMS) can be used to self-manage Customer Master Keys (CMK). Through ElastiCache integration with AWS Key Management Service, you are able to create, own, and manage the keys used for encryption of data at rest for your ElastiCache cluster.

  **[Resources]:**

  - Using customer managed keys from AWS Key Management Service
  - AWS Key Management Service
  - AWS KMS concepts

## SEC 5: How do you encrypt in-transit data with ElastiCache?

**Question-level introduction:** It is a common requirement to mitigate against data being compromised while in transit. This represents data within components of a distributed system, as well as between application clients and cluster nodes. ElastiCache supports this requirement by allowing for encrypting data in-transit between clients and cluster, and between cluster nodes themselves. Instance types in the M6g and R6g families also feature always-on, in-memory encryption.

**Question-level benefit:** Amazon ElastiCache in-transit encryption is an optional feature that allows you to increase the security of your data at its most vulnerable points, when it is in-transit from one location to another.

- **[Required]** In-transit encryption can only be enabled on a cluster (replication group) upon creation. Please note that, due to the additional processing required for encrypting/decrypting data, implementing in-transit encryption will have some performance impact. To understand the impact, it is recommended to benchmark your workload before and after enabling encryption-in-transit.

  **[Resources]:**

  - In-transit encryption overview

## SEC 6: How do you restrict access to control plane resources?

**Question-level introduction:** IAM policies and ARN enable fine grained access controls for ElastiCache for Valkey and Redis OSS, allowing for tighter control to manage the creation, modification and deletion of clusters.

**Question-level benefit:** Management of Amazon ElastiCache resources, such as replication groups, nodes, etc. can be constrained to AWS accounts that have specific permissions based on IAM policies, improving security and reliability of resources.

- **[Required]** Manage access to Amazon ElastiCache resources by assigning specific AWS Identity and Access Managementpolicies to AWS users, allowing finer control over which accounts can perform what actions on clusters.

  **[Resources]:**

  - Overview of managing access permissions to your ElastiCache resources
  - Using identity-based policies (IAM policies) for Amazon ElastiCache

## SEC 7: How do you detect and respond to security events?

**Question-level introduction:** ElastiCache, when deployed with RBAC enabled, exports CloudWatch metrics to notify users of security events. These metrics help identify failed attempts to authenticate, access keys, or run commands that connecting RBAC users are not authorized for.

Additionally, AWS products and services resources help secure your overall workload by automating deployments and logging all actions and modifications for later review/audit.

**Question-level benefit:** By monitoring events, you enable your organization to respond according to your requirements, policies, and procedures. Automating the monitoring and responses to these security events hardens your overall security posture.

- **[Required]** Familiarize yourself with the CloudWatch Metrics published that pertain to RBAC authentication and authorization failures.
  - AuthenticationFailures = Failed attempts to authenticate to Valkey or Redis OSS
  - KeyAuthorizationFailures = Failed attempts by users to access keys without permission
  - CommandAuthorizationFailures = Failed attempts by users to run commands without permission

**[Resources]:**

- Metrics for Valkey or Redis OSS

- **[Best]** It is recommended to setup alerts and notifications on these metrics and respond as necessary.

  **[Resources]:**

  - Using Amazon CloudWatch alarms

- **[Best]** Use the Valkey or Redis OSS ACL LOG command to gather further details

  **[Resources]:**

  - ACL LOG

- **[Best]** Familiarize yourself with the AWS products and services capabilities as it pertains to monitoring, logging, and analyzing ElastiCache deployments and events

  **[Resources]:**

  - Logging Amazon ElastiCache API calls with AWS CloudTrail
  - elasticache-redis-cluster-automatic-backup-check
  - Monitoring use with CloudWatch Metrics

## Amazon ElastiCache Well-Architected Lens Reliability Pillar

The reliability pillar focuses on workloads performing their intended functions and how to recover quickly from failure to meet demands. Key topics include distributed system design, recovery planning, and adapting to changing requirements.

**Topics**

- REL 1: How are you supporting high availability (HA) architecture deployments?
- REL 2: How are you meeting your Recovery Point Objectives (RPOs) with ElastiCache?
- REL 3: How do you support disaster recovery (DR) requirements?
- REL 4: How do you effectively plan for failovers?
- REL 5: Are your ElastiCache components designed to scale?

## REL 1: How are you supporting high availability (HA) architecture deployments?

**Question-level introduction:** Understanding the high availability architecture of Amazon ElastiCache will enable you to operate in a resilient state during availability events.

**Question-level benefit:** Architecting your ElastiCache clusters to be resilient to failures ensures higher availability for your ElastiCache deployments.

- **[Required]** Determine the level of reliability you require for your ElastiCache cluster. Different workloads have different resiliency standards, from entirely ephemeral to mission critical workloads. Define needs for each type of environment you operate such as dev, test, and production.

  Caching engine: ElastiCache for Memcached vs ElastiCache for Valkey and Redis OSS

  1. ElastiCache for Memcached does not provide any replication mechanism and is used primarily for ephemeral workloads.
  2. ElastiCache for Valkey and Redis OSS offers HA features discussed below
- **[Best]** For workloads that require HA, use ElastiCache in cluster mode with a minimum of two replicas per shard, even for small throughput requirement workloads that require only one shard.
  1. For cluster mode enabled, multi-AZ is enabled automatically.

     Multi-AZ minimizes downtime by performing automatic failovers from primary node to replicas, in case of any planned or unplanned maintenance as well as mitigating AZ failure.
  2. For sharded workloads, a minimum of three shards provides faster recovery during failover events as the Valkey or Redis OSS Cluster Protocol requires a majority of primary nodes be available to achieve quorum.
  3. Set up two or more replicas across Availability.

     Having two replicas provides improved read scalability and also read availability in scenarios where one replica is undergoing maintenance.
  4. Use Graviton2-based node types (default nodes in most regions).

     ElastiCache has added optimized performance on these nodes. As a result, you get better replication and synchronization performance, resulting in overall improved availability.
  5. Monitor and right-size to deal with anticipated traffic peaks: under heavy load, the engine may become unresponsive, which affects availability. `BytesUsedForCache` and `DatabaseMemoryUsagePercentage` are good indicators of your memory usage, whereas

`ReplicationLag` is an indicator of your replication health based on your write rate. You can use these metrics to trigger cluster scaling.

6. Ensure client-side resiliency by testing with the Failover API prior to a production failover event.

**[Resources]:**

- Configure ElastiCache for Redis OSS for higher availability
- High availability using replication groups

## REL 2: How are you meeting your Recovery Point Objectives (RPOs) with ElastiCache?

**Question-level introduction:** Understand workload RPO to inform decisions on ElastiCache backup and recovery strategies.

**Question-level benefit:** Having an in-place RPO strategy can improve business continuity in the event of a disaster recovery scenarios. Designing your backup and restore policies can help you meet your Recovery Point Objectives (RPO) for your ElastiCache data. ElastiCache offers snapshot capabilities which are stored in Amazon S3, along with a configurable retention policy. These snapshots are taken during a defined backup window, and handled by the service automatically. If your workload requires additional backup granularity, you have the option to create up to 20 manual backups per day. Manually created backups do not have a service retention policy and can be kept indefinitely.

- **[Required]** Understand and document the RPO of your ElastiCache deployments.
  - Be aware that Memcached does not offer any backup processes.
  - Review the capabilities of ElastiCache Backup and Restore features.
- **[Best]** Have a well-communicated process in place for backing up your cluster.
  - Initiate manual backups on an as-needed basis.
  - Review retention policies for automatic backups.
  - Note that manual backups will be retained indefinitely.
  - Schedule your automatic backups during periods of low usage.
  - Perform backup operations against read-replicas to ensure you minimize the impact on cluster performance.

- **[Good]** Leverage the scheduled backup feature of ElastiCache to regularly back up your data during a defined window.
  - Periodically test restores from your backups.
- **[Resources]:**
  - Redis OSS
  - Backup and restore for ElastiCache
  - Making manual backups
  - Scheduling automatic backups
  - Backup and Restore ElastiCache Clusters

## REL 3: How do you support disaster recovery (DR) requirements?

**Question-level introduction:** Disaster recovery is an important aspect of any workload planning. ElastiCache offers several options to implement disaster recovery based on workload resilience requirements. With Amazon ElastiCache Global Datastore, you can write to your cluster in one Region and have the data available to be read from two other cross-Region replica clusters, thereby enabling low-latency reads and disaster recovery across regions.

**Question-level benefit:** Understanding and planning for a variety of disaster scenarios can ensure business continuity. DR strategies must be balanced against cost, performance impact, and data loss potential.

- **[Required]** Develop and document DR strategies for all your ElastiCache components based upon workload requirements. ElastiCache is unique in that some use cases are entirely ephemeral and don't require any DR strategy, whereas others are on the opposite end of the spectrum and require an extremely robust DR strategy. All options must be weighed against Cost Optimization – greater resiliency requires larger amounts of instrastructure.

  Understand the DR options available on a regional and multi-region level.
  - Multi-AZ Deployments are recommended to guard against AZ failure. Be sure to deploy with Cluster-Mode enabled in Multi-AZ architectures, with a minimum of 3 AZs available.
  - Global Datastore is recommended to guard against regional failures.
- **[Best]** Enable Global Datastore for workloads that require region level resiliency.
  - Have a plan to failover to secondary region in case of primary degradation.
  - Test multi-region failover process prior to a failover over in production.

- Monitor `ReplicationLag` metric to understand potential impact of data loss during failover events.
- **[Resources]:**
  - Mitigating Failures
  - Replication across AWS Regions using global datastores
  - Restoring from a backup with optional cluster resizing
  - Minimizing downtime in ElastiCache for Valkey and Redis OSS with Multi-AZ

## REL 4: How do you effectively plan for failovers?

**Question-level introduction:** Enabling multi-AZ with automatic failovers is an ElastiCache best practice. In certain cases, ElastiCache for Valkey and Redis OSS replaces primary nodes as part of service operations. Examples include planned maintenance events and the unlikely case of a node failure or availability zone issue. Successful failovers rely on both ElastiCache and your client library configuration.

**Question-level benefit:** Following best practices for ElastiCache failovers in conjunction with your specific ElastiCache client library helps you minimize potential downtime during failover events.

- **[Required]** For cluster mode disabled, use timeouts so your clients detect if it needs to disconnect from the old primary node and reconnect to the new primary node, using the updated primary endpoint IP address. For cluster mode enabled, the client library is responsible with detecting changes in the underlying cluster topology. This is accomplished most often by configuration settings in the ElastiCache client library, which also allow you to configure the frequency and the method of refresh. Each client library offers its own settings and more details are available in their corresponding documentation.

  **[Resources]:**
  - Minimizing downtime in ElastiCache for Valkey and Redis OSS with Multi-AZ
  - Review the best practices of your ElastiCache client library.
- **[Required]** Successful failovers depend on a healthy replication environment between the primary and the replica nodes. Review and understand the asynchronous nature of Valkey and Redis OSS replication, as well as the available CloudWatch metrics to report on the replication lag between primary and replica nodes. For use cases that require greater data safety, leverage the WAIT command to force replicas to acknowledge writes before responding to connected clients.

**[Resources]:**

- Metrics for Valkey or Redis OSS
- Monitoring best practices with ElastiCache using Amazon CloudWatch

- **[Best]** Regularly validate the responsiveness of your application during failover using the ElastiCache Test Failover API.

  **[Resources]:**

  - Testing Automatic Failover to a Read Replica on ElastiCache
  - Testing automatic failover

## REL 5: Are your ElastiCache components designed to scale?

**Question-level introduction:** By understanding the scaling capabilities and available deployment topologies, your ElastiCache components can adjust over time to meet changing workload requirements. ElastiCache offers 4-way scaling: in/out (horizontal) as well as up/down (vertical).

**Question-level benefit:** Following best practices for ElastiCache deployments provides the greatest amount of scaling flexibility, as well as meeting the Well Architected principle of scaling horizontally to minimize the impact of failures.

- **[Required]** Understand the difference between Cluster-mode Enabled and Cluster-mode Disabled topologies. In almost all cases it is recommended to deploy with Cluster-mode enabled as it allow for greater scalability over time. Cluster-mode disabled components are limited in their ability to horizontally scale by adding read replicas.
- **[Required]** Understand when and how to scale.
  - For more READIOPS: add replicas
  - For more WRITEOPS: add shards (scale out)
  - For more network IO – use network optimized instances, scale up
- **[Best]** Deploy your ElastiCache components with Cluster-mode enabled, with a bias toward more, smaller nodes rather than fewer, larger nodes. This effectively limits the blast radius of a node failure.
- **[Best]** Include replicas in your clusters for enhanced responsiveness during scaling events
- **[Good]** For cluster-mode disabled, leverage read replicas to increase overall read capacity. ElastiCache has support for up to 5 read replicas in cluster-mode disabled, as well as vertical scaling.

- **[Resources]:**
  - Scaling ElastiCache clusters
  - Online scaling up

# Amazon ElastiCache Well-Architected Lens Performance Efficiency Pillar

The performance efficiency pillar focuses on using IT and computing resources efficiently. Key topics include selecting the right resource types and sizes based on workload requirements, monitoring performance, and making informed decisions to maintain efficiency as business needs evolve.

**Topics**


- PE 1: How do you monitor the performance of your Amazon ElastiCache cluster?
- PE 2: How are you distributing work across your ElastiCache Cluster nodes?
- PE 3: For caching workloads, how do you track and report the effectiveness and performance of your cache?
- PE 4: How does your workload optimize the use of networking resources and connections?
- PE 5: How do you manage key deletion and/or eviction?
- PE 6: How do you model and interact with data in ElastiCache?
- PE 7: How do you log slow running commands in your Amazon ElastiCache cluster?
- PE8: How does Auto Scaling help in increasing the performance of the ElastiCache cluster?


## PE 1: How do you monitor the performance of your Amazon ElastiCache cluster?

**Question-level introduction:** By understanding the existing monitoring metrics, you can identify current utilization. Proper monitoring can help identify potential bottlenecks impacting the performance of your cluster.

**Question-level benefit:** Understanding the metrics associated with your cluster can help guide optimization techniques that can lead to reduced latency and increased throughput.

- **[Required]** Baseline performance testing using a subset of your workload.
  - You should monitor performance of the actual workload using mechanisms such as load testing.

  - Monitor the CloudWatch metrics while running these tests to gain an understanding of metrics available, and to establish a performance baseline.
- **[Best]** For ElastiCache for Valkey and Redis OSS workloads, rename computationally expensive commands, such as KEYS, to limit the ability of users to run blocking commands on production clusters.
  - ElastiCache workloads running engine 6.x for Redis OSS can leverage role-based access control to restrict certain commands. Access to the commands can be controlled by creating Users and User Groups with the AWS Console or CLI, and associating the User Groups to a cluster. In Redis OSS 6, when RBAC is enabled, we can use "-@dangerous" and it will disallow expensive commands like KEYS, MONITOR, SORT, etc. for that user.
  - For engine version 5.x, rename commands using the `rename-commands` parameter on the cluster parameter group.
- **[Better]** Analyze slow queries and look for optimization techniques.
  - For ElastiCache for Valkey and Redis OSS workloads, learn more about your queries by analyzing the Slow Log. For example, you can use the following command, `valkey-cli slowlog get 10` to show last 10 commands which exceeded latency thresholds (10 milliseconds by default).
  - Certain queries can be performed more efficiently using complex ElastiCache for Valkey and Redis OSS data structures. As an example, for numerical style range lookups, an application can implement simple numerical indexes with Sorted Sets. Managing these indexes can reduce scans performed on the data set, and return data with greater performance efficiency.
  - For ElastiCache for Valkey and Redis OSS workloads, `redis-benchmark` provides a simple interface for testing the performance of different commands using user defined inputs like number of clients, and size of data.
  - Since Memcached only supports simple key level commands, consider building additional keys as indexes to avoid iterating through the key space to serve client queries.
- **[Resources]:**
  - Monitoring use with CloudWatch Metrics
  - Using Amazon CloudWatch alarms
  - Valkey and Redis OSS specific parameters
  - SLOWLOG
  - benchmark

## PE 2: How are you distributing work across your ElastiCache Cluster nodes?

**Question-level introduction:** The way your application connects to Amazon ElastiCache nodes can impact the performance and scalability of the cluster.

**Question-level benefit:** Making proper use of the available nodes in the cluster will ensure that work is distributed across the available resources. The following techniques help avoid idle resources as well.

- **[Required]** Have clients connect to the proper ElastiCache endpoint.
  - ElastiCache for Valkey and Redis OSS implements different endpoints based on the cluster mode in use. For cluster mode enabled, ElastiCache will provide a configuration endpoint. For cluster mode disabled, ElastiCache provides a primary endpoint, typically used for writes, and a reader endpoint for balancing reads across replicas. Implementing these endpoints correctly will results in better performance, and easier scaling operations. Avoid connecting to individual node endpoints unless there is a specific requirement to do so.
  - For multi-node Memcached clusters, ElastiCache provides a configuration endpoint which enables Auto Discovery. It is recommended to use a hashing algorithm to distribute work evenly across the cache nodes. Many Memcached client libraries implement consistent hashing. Check the documentation for the library you are using to see if it supports consistent hashing and how to implement it. You can find more information on implementing these features here.
- **[Better]** Take advantage of ElastiCache for Valkey and Redis OSS cluster mode enabled clusters to improve scalability.
  - ElastiCache for Valkey and Redis OSS (cluster mode enabled) clusters support online scaling operations (out/in and up/down) to help distribute data dynamically across shards. Using the Configuration Endpoint will ensure your cluster aware clients can adjust to changes in the cluster topology.
  - You may also rebalance the cluster by moving hashslots between available shards in your ElastiCache for Valkey and Redis OSS (cluster mode enabled) cluster. This helps distribute work more efficiently across available shards.
- **[Better]** Implement a strategy for identifying and remediating hot keys in your workload.
  - Consider the impact of multi-dimensional Valkey or Redis OSS data structures such a lists, streams, sets, etc. These data structures are stored in single Keys, which reside on a single node. A very large multi-dimensional key has the potential to utilize more network capacity

and memory than other data types and can cause a disproportionate use of that node. If possible, design your workload to spread out data access across many discrete Keys.

- Hot keys in the workload can impact performance of the node in use. For ElastiCache for Valkey and Redis OSS workloads, you can detect hot keys using `valkey-cli --hotkeys` if an LFU max-memory policy is in place.
- Consider replicating hot keys across multiple nodes to distribute access to them more evenly. This approach requires the client to write to multiple primary nodes (the Valkey or Redis OSS node itself will not provide this functionality) and to maintain a list of key names to read from, in addition to the original key name.
- ElastiCache engine 7.2 for Valkey and above, and ElastiCache version 6 for Redis OSS and above, all support server-assisted client-side caching. This enables applications to wait for changes to a key before making network calls back to ElastiCache.

- **[Resources]:**
  - Configure ElastiCache for Valkey and Redis OSS for higher availability
  - Finding connection endpoints in ElastiCache
  - Load balancing best practices
  - Online resharding for Valkey or Redis OSS (cluster mode enabled)
  - Client-side caching in Valkey and Redis OSS

## PE 3: For caching workloads, how do you track and report the effectiveness and performance of your cache?

**Question-level introduction:** Caching is a commonly encountered workload on ElastiCache and it is important that you understand how to manage the effectiveness and performance of your cache.

**Question-level benefit:** Your application may show signs of sluggish performance. Your ability to use cache specific metrics to inform your decision on how to increase app performance is critical for your cache workload.

- **[Required]** Measure and track over time the cache-hits ratio. The efficiency of your cache is determined by its 'cache hits ratio'. The cache hits ratio is defined by the total of key hits divided by the total hits and misses. The closer to 1 the ratio is, the more effective your cache is. A low cache hits ratio is caused by the volume of cache misses. Cache misses occur when the requested key is not found in the cache. A key is not in the cache because it either has been evicted or

deleted, has expired, or has never existed. Understand why keys are not in cache and develop appropriate strategies to have them in cache.

**[Resources]:**

- Metrics for Valkey and Redis OSS

- **[Required]** Measure and collect your application cache performance in conjunction with latency and CPU utilization values to understand whether you need to make adjustments to your time-to-live or other application components. ElastiCache provides a set of CloudWatch metrics for aggregated latencies for each data structure. These latency metrics are calculated using the commandstats statistic from the INFO command and do not include the network and I/O time. This is only the time consumed by ElastiCache to process the operations.

  **[Resources]:**

  - Metrics for Valkey and Redis OSS
  - Monitoring best practices with ElastiCache using Amazon CloudWatch

- **[Best]** Choose the right caching strategy for your needs. A low cache hits ratio is caused by the volume of cache misses. If your workload is designed to have low volume of cache misses (such as real time communication), it is best to conduct reviews of your caching strategies and apply the most appropriate resolutions for your workload, such as query instrumentation to measure memory and performance. The actual strategies you use to implement for populating and maintaining your cache depend on what data your clients need to cache and the access patterns to that data. For example, it is unlikely that you will use the same strategy for both personalized recommendations on a streaming application, and for trending news stories.

  **[Resources]:**

  - Caching strategies for Memcached
  - Caching Best Practices
  - Performance at Scale with Amazon ElastiCache Whitepaper

## PE 4: How does your workload optimize the use of networking resources and connections?

**Question-level introduction:** ElastiCache for Valkey, Memcached, and Redis OSS are supported by many application clients, and implementations may vary. You need to understand the networking and connection management in place to analyze potential performance impact.

**Question-level benefit:** Efficient use of networking resources can improve the performance efficiency of your cluster. The following recommendations can reduce networking demands, and improve cluster latency and throughput.

- **[Required]** Proactively manage connections to your ElastiCache cluster.
  - Connection pooling in the application reduces the amount of overhead on the cluster created by opening and closing connections. Monitor connection behavior in Amazon CloudWatch using `CurrConnections` and `NewConnections`.
  - Avoid connection leaking by properly closing client connections where appropriate. Connection management strategies include properly closing connections that are not in use, and setting connection time-outs.
  - For Memcached workloads, there is a configurable amount of memory reserved for handling connections called, `memcached_connections_overhead`.
- **[Better]** Compress large objects to reduce memory, and improve network throughput.
  - Data compression can reduce the amount of network throughput required (Gbps), but increases the amount of work on the application to compress and decompress data.
  - Compression also reduces the amount of memory consumed by keys
  - Based on your application needs, consider the trade-offs between compression ratio and compression speed.
- **[Resources]:**
  - ElastiCache - Global Datastore
  - Memcached specific parameters
  - ElastiCache version 5.0.3 for Redis OSS enhances I/O handling to boost performance
  - Metrics for Valkey and Redis OSS
  - Configure ElastiCache for higher availability

## PE 5: How do you manage key deletion and/or eviction?

**Question-level introduction:** Workloads have different requirements, and expected behavior when a cluster node is approaching memory consumption limits. ElastiCache has different policies for handling these situations.

**Question-level benefit:** Proper management of available memory, and understanding of eviction policies will help ensure awareness of cluster behavior when instance memory limits are exceeded.

- **[Required]** Instrument the data access to evaluate which policy to apply. Identify an appropriate max-memory policy to control if and how evictions are performed on the cluster.
  - Eviction occurs when the max-memory on the cluster is consumed and a policy is in place to allow eviction. The behavior of the cluster in this situation depends on the eviction policy specified. This policy can be managed using the `maxmemory-policy` on the cluster parameter group.
  - The default policy `volatile-lru` frees up memory by evicting keys with a set expiration time (TTL value). Least frequently used (LFU) and least recently used (LRU) policies remove keys based on usage.
  - For Memcached workloads, there is a default LRU policy in place controlling evictions on each node. The number of evictions on your Amazon ElastiCache cluster can be monitored using the Evictions metric on Amazon CloudWatch.
- **[Better]** Standardize delete behavior to control performance impact on your cluster to avoid unexpected performance bottlenecks.
  - For ElastiCache for Valkey and Redis OSS workloads, the `UNLINK` command behaves like `DEL` in that it removes the specified keys. However, `UNLINK` reclaims memory in a separate thread, so it is non-blocking, although `DEL` is blocking. With `UNLINK`, the actual memory reclamation occurs asynchronously, after the command returns.
  - For ElastiCache version 6.x for Redis OSS workloads, the behavior of the `DEL` command can be modified in the parameter group using `lazyfree-lazy-user-del` parameter.
- **[Resources]:**
  - Configuring engine parameters using ElastiCache parameter groups
  - UNLINK
  - Cloud Financial Management with AWS

## PE 6: How do you model and interact with data in ElastiCache?

**Question-level introduction:** ElastiCache is heavily application dependent on the data structures and the data model used, but it also needs to consider the underlying data store (if present). Understand the data structures available and ensure you are using the most appropriate data structures for your needs.

**Question-level benefit:** Data modeling in ElastiCache has several layers, including application use case, data types, and relationships between data elements. Additionally, each data type and command have their own well documented performance signatures.

- **[Best]** A best practice is to reduce unintentional overwriting of data. Use a naming convention that minimizes overlapping key names. Conventional naming of your data structures uses a hierarchical method such as: `APPNAME:CONTEXT:ID`, such as `ORDER-APP:CUSTOMER:123`.

  **[Resources]:**

  - Key naming
- **[Best]** ElastiCache for Valkey and Redis OSS commands have a time complexity defined by the Big O notation. This time complexity of a command is a algorithmic/mathematical representation of its impact. When introducing a new data type in your application you need to carefully review the time complexity of the related commands. Commands with a time complexity of O(1) are constant in time and do not depend on the size of the input however commands with a time complexity of O(N) are linear in time and are subject to the size of the input. Due to the single threaded design of ElastiCache for Valkey and Redis OSS, large volume of high time complexity operations will result in lower performance and potential operation timeouts.

  **[Resources]:**

  - Commands
- **[Best]** Use APIs to gain GUI visibility into the data model in your cluster.

  **[Resources]:**

  - Redis OSS Commander
  - Redis OSS Browser
  - Redsmin

## PE 7: How do you log slow running commands in your Amazon ElastiCache cluster?

**Question-level introduction:** Performance tuning benefits through the capture, aggregation, and notification of long-running commands. By understanding how long it takes for commands to execute, you can determine which commands result in poor performance as well as commands that block the engine from performing optimally. ElastiCache also has the capability to forward this information to Amazon CloudWatch or Amazon Kinesis Data Firehose.

**Question-level benefit:** Logging to a dedicated permanent location and providing notification events for slow commands can help with detailed performance analysis and can be used to trigger automated events.

- **[Required]** ElastiCache running a Valkey engine 7.2 or newer, or running a Redis OSS engine version 6.0 or newer, properly configured parameter group and SLOWLOG logging enabled on the cluster.
  - The required parameters are only available when engine version compatibility is set to Valkey 7.2 and higher, or Redis OSS version 6.0 or higher.
  - SLOWLOG logging occurs when the server execution time of a command takes longer than a specified value. The behavior of the cluster depends on the associated Parameter Group parameters which are `slowlog-log-slower-than` and `slowlog-max-len`.
  - Changes take effect immediately.
- **[Best]** Take advantage of CloudWatch or Kinesis Data Firehose capabilities.
  - Use the filtering and alarm capabilities of CloudWatch, CloudWatch Logs Insights and Amazon Simple Notification Services to achieve performance monitoring and event notification.
  - Use the streaming capabilities of Kinesis Data Firehose to archive SLOWLOG logs to permanent storage or to trigger automated cluster parameter tuning.
  - Determine if JSON or plain TEXT format suits your needs best.
  - Provide IAM permissions to publish to CloudWatch or Kinesis Data Firehose.
- **[Better]** Configure `slowlog-log-slower-than` to a value other than the default.
  - This parameter determines how long a command may execute for within the Valkey or Redis OSS engine before it is logged as a slow running command. The default value is 10,000 microseconds (10 milliseconds). The default value may be too high for some workloads.
  - Determine a value that is more appropriate for your workload based on application needs and testing results; however, a value that is too low may generate excessive data.
- **[Better]** Leave `slowlog-max-len` at the default value.
  - This parameter determines the upper limit for how many slow-running commands are captured in Valkey or Redis OSS memory at any given time. A value of 0 effectively disables the capture. The higher the value, the more entries will be stored in memory, reducing the chance of important information being evicted before it can be reviewed. The default value is 128.

- The default value is appropriate for most workloads. If there is a need to analyze data in an expanded time window from the valkey-cli via the SLOWLOG command, consider increasing this value. This allows more commands to remain in Valkey or Redis OSS memory.

  If you are emitting the SLOWLOG data to either CloudWatch Logs or Kinesis Data Firehose, the data will be persisted and can be analyzed outside of the ElastiCache system, reducing the need to store large numbers of slow running commands in Valkey or Redis OSS memory.

- **[Resources]:**
  - How do I turn on Slow log in a cluster?
  - Log delivery
  - Redis OSS-specific parameters
  - https://aws.amazon.com/cloudwatch/Amazon CloudWatch
  - Amazon Kinesis Data Firehose

## PE8: How does Auto Scaling help in increasing the performance of the ElastiCache cluster?

**Question-level introduction:** By implementing the feature of Valkey or Redis OSS auto scaling, your ElastiCache components can adjust over time to increase or decrease the desired shards or replicas automatically. This can be done by implementing either the target tracking or scheduled scaling policy.

**Question-level benefit:** Understanding and planning for the spikes in the workload can ensure enhanced caching performance and business continuity. ElastiCache Auto Scaling continually monitors your CPU/Memory utilization to make sure your cluster is operating at your desired performance levels.

- **[Required]** When launching a cluster for ElastiCache for Valkey or Redis OSS:
  1. Ensure that the Cluster mode is enabled
  2. Make sure the instance belongs to a family of certain type and size that support auto scaling
  3. Ensure the cluster is not running in Global datastores, Outposts or Local Zones

  **[Resources]:**

  - Scaling clusters in Valkey and Redis OSS (Cluster Mode Enabled)
  - Using Auto Scaling with shards

  - Using Auto Scaling with replicas
- **[Best]** Identify if your workload is read-heavy or write-heavy to define scaling policy. For best performance, use just one tracking metric. It is recommended to avoid multiple policies for each dimension, as auto scaling policies scale out when the target is hit, but scale in only when all target tracking policies are ready to scale in.

  **[Resources]:**

  - Auto Scaling policies
  - Defining a scaling policy
- **[Best]** Monitoring performance over time can help you detect workload changes that would remain unnoticed if monitored at a particular point in time. You can analyze corresponding CloudWatch metrics for cluster utilization over a four-week period to determine the target value threshold. If you are still not sure of what value to choose, we recommend starting with a minimum supported predefined metric value.

  **[Resources]:**

  - Monitoring use with CloudWatch Metrics
- **[Better]** We advise testing your application with expected minimum and maximum workloads, to identify the exact number of shards/replicas required for the cluster to develop scaling policies and mitigate availability issues.

  **[Resources]:**

  - Registering a Scalable Target
  - Registering a Scalable Target using the AWS CLI

## Amazon ElastiCache Well-Architected Lens Cost Optimization Pillar

The cost optimization pillar focuses on avoiding unnecessary costs. Key topics include understanding and controlling where money is being spent, selecting the most appropriate node type (use instances that support data tiering based on workload needs), the right number of resource types (how many read replicas) , analyzing spend over time, and scaling to meet business needs without overspending.

**Topics**

- COST 1: How do you identify and track costs associated with your ElastiCache resources? How do you develop mechanisms to enable users to create, manage, and dispose of created resources?

- COST 2: How do you use continuous monitoring tools to help you optimize the costs associated with your ElastiCache resources?
- COST 3: Should you use an instance type that support data tiering? What are the advantages of a data tiering? When not to use data tiering instances?

## COST 1: How do you identify and track costs associated with your ElastiCache resources? How do you develop mechanisms to enable users to create, manage, and dispose of created resources?

**Question-level introduction:** Understanding cost metrics requires the participation of and collaboration across multiple teams: software engineering, data management, product owners, finance, and leadership. Identifying key cost drivers requires all involved parties understand service usage control levers and cost management trade-offs and it is frequently the key difference between successful and less successful cost optimization efforts. Ensuring you have processes and tools in place to track resources created from development to production and retirement helps you manage the costs associated with ElastiCache.

**Question-level benefit:** Continuous tracking of all costs associated with your workload requires a deep understanding of the architecture that includes ElastiCache as one of its components. Additionally, you should have a cost management plan in place to collect and compare usage against your budget.

- **[Required]** Institute a Cloud Center of Excellence (CCoE) with one of its founding charters to own defining, tracking, and taking action on metrics around your organizations' ElastiCache usage. If a CCoE exists and functions, ensure that it knows how to read and track costs associated with ElastiCache. When resources are created, use IAM roles and policies to validate that only specific teams and groups can instantiate resources. This ensures that costs are associated with business outcomes and a clear line of accountability is established, from a cost perspective.
  1. CCoE should identify, define, and publish cost metrics that are updated on a regular -monthly- basis around key ElastiCache usage across categorical data such as:
     a. Types of nodes used and their attributes: standard vs. memory optimized, on-demand vs. reserved instances, regions and availability zones
     b. Types of environments: free, dev, testing, and production
     c. Backup storage and retention strategies
     d. Data transfer within and across regions
     e. Instances running on Amazon Outposts

2. CCoE consists of a cross-functional team with non-exclusive representation from software engineering, data management, product team, finance, and leadership teams in your organization.

**[Resources]:**

- Create a Cloud Center of Excellence
- Amazon ElastiCache pricing

- **[Required]** Use cost allocation tags to track costs at a low level of granularity. Use AWS Cost Management to visualize, understand, and manage your AWS costs and usage over time.

  1. Use tags to organize your resources, and cost allocation tags to track your AWS costs on a detailed level. After you activate cost allocation tags, AWS uses the cost allocation tags to organize your resource costs on your cost allocation report, to make it easier for you to categorize and track your AWS costs. AWS provides two types of cost allocation tags, an AWS generated tags and user-defined tags. AWS defines, creates, and applies the AWS generated tags for you, and you define, create, and apply user-defined tags. You must activate both types of tags separately before they can appear in Cost Management or on a cost allocation report.
  2. Use cost allocation tags to organize your AWS bill to reflect your own cost structure. When you add cost allocation tags to your resources in Amazon ElastiCache, you will be able to track costs by grouping expenses on your invoices by resource tag values. You should consider combining tags to track costs at a greater level of detail.

  **[Resources]:**

  - Using AWS cost allocation tags
  - Monitoring costs with cost allocation tags
  - AWS Cost Explorer

- **[Best]** Connect ElastiCache cost to metrics that reach across the organization.

  1. Consider business metrics as well as operational metrics like latency - what concepts in your business model are understandable across roles? The metrics need to be understandable by as many roles as possible in the organization.
  2. Examples - simultaneous served users, max and average latency per operation and user, user engagement scores, user return rates/week, session length/user, abandonment rate, cache hit rate, and keys tracked

  **[Resources]:**

- Monitoring use with CloudWatch Metrics

- **[Good]** Maintain up-to-date architectural and operational visibility on metrics and costs across the entire workload that uses ElastiCache.

  1. Understand your entire solution ecosystem, ElastiCache tends to be part of a full ecosystem of AWS services in their technology set, from clients to API Gateway, Redshift, and QuickSight for reporting tools (for example).
  2. Map components of your solution from clients, connections, security, in-memory operations, storage, resource automation, data access and management, on your architecture diagram. Each layer connects to the entire solution and has its own needs and capabilities that add to and/or help you manage the overall cost.
  3. Your diagram should include the use of compute, networking, storage, lifecycle policies, metrics gathering as well as the operational and functional ElastiCache elements of your application
  4. The requirements of your workload are likely to evolve over time and it is essential that you continue to maintain and document your understanding of the underlying components as well as your primary functional objectives in order to remain proactive in your workload cost management.
  5. Executive support for visibility, accountability, prioritization, and resources is crucial to you having an effective cost management strategy for your ElastiCache.

## COST 2: How do you use continuous monitoring tools to help you optimize the costs associated with your ElastiCache resources?

**Question-level introduction:** You need to aim for a proper balance between your ElastiCache cost and application performance metrics. Amazon CloudWatch provides visibility into key operational metrics that can help you assess whether your ElastiCache resources are over or under utilized, relative to your needs. From a cost optimization perspective, you need to understand when you are overprovisioned and be able to develop appropriate mechanisms to resize your ElastiCache resources while maintaining your operational, availability, resilience, and performance needs.

**Question-level benefit:** In an ideal state, you will have provisioned sufficient resources to meet your workload operational needs and not have under-utilized resources that can lead to a sub-optimal cost state. You need to be able to both identify and avoid operating oversized ElastiCache resources for long periods of time.

- **[Required]** Use CloudWatch to monitor your ElastiCache clusters and analyze how these metrics relate to your AWS Cost Explorer dashboards.

  1. ElastiCache provides both host-level metrics (for example, CPU usage) and metrics that are specific to the cache engine software (for example, cache gets and cache misses). These metrics are measured and published for each cache node in 60-second intervals.
  2. ElastiCache performance metrics (CPUUtilization, EngineUtilization, SwapUsage, CurrConnections, and Evictions) may indicate that you need to scale up/down (use larger/smaller cache node types) or in/out (add more/less shards). Understand the cost implications of scaling decisions by creating a playbook matrix that estimates the additional cost and the min and max lengths of time required to meet your application performance thresholds.

  **[Resources]:**

  - Monitoring use with CloudWatch Metrics
  - Which Metrics Should I Monitor?
  - Amazon ElastiCache pricing

- **[Required]** Understand and document your backup strategy and cost implications.

  1. With ElastiCache, the backups are stored in Amazon S3, which provides durable storage. You need to understand the cost implications in relation to your ability to recover from failures.
  2. Enable automatic backups that will delete backup files that are past the retention limit.

  **[Resources]:**

  - Scheduling automatic backups
  - Amazon Simple Storage Service pricing

- **[Best]** Use Reserved Nodes for your instances as a deliberate strategy to manage costs for workloads that are well understood and documented. Reserved nodes are charged an up front fee that depends upon the node type and the length of reservation—one or three years. This charge is much less than the hourly usage charge that you incur with On-Demand nodes.

  1. You may need to operate your ElastiCache clusters using on-demand nodes until you have gathered sufficient data to estimate the reserved instance requirements. Plan and document the resources needed to meet your needs and compare expected costs across instance types (on-demand vs. reserved)
  2. Regularly evaluate new cache node types available and assess whether it makes sense, from a cost and operational metrics perspective, to migrate your instance fleet to new cache node types

## COST 3: Should you use an instance type that support data tiering? What are the advantages of a data tiering? When not to use data tiering instances?

**Question-level introduction:** Selecting the appropriate instance type can not only have performance and service level impact but also financial impact. Instance types have different cost associated with them. Selecting one or a few large instance types that can accommodate all storage needs in memory might be a natural decision. However, this could have significant cost impact as the project matures. Ensuring that the correct instance type is selected requires periodic examination of ElastiCache object idle time.

**Question-level benefit:** You should have a clear understanding of how various instance types impact your cost at the present and in the future. Marginal or periodic workload changes should not cause disproportionate costs changes. If the workload permits it, instance types that support data tiering offer a better price per storage available storage. Because of the per instance available SSD storage data tiering instances support a much higher total data per instance capability.

- **[Required]** Understand limitations of data tiering instances
  1. Only available for ElastiCache for Valkey or Redis OSS clusters.
  2. Only limited instance types support data tiering.
  3. Only ElastiCache version 6.2 for Redis OSS and above is supported
  4. Large items are not swapped out to SSD. Objects over 128 MiB are kept in memory.

  **[Resources]:**

  - Data tiering
  - Amazon ElastiCache pricing
- **[Required]** Understand what percentage of your database is regularly accessed by your workload.
  1. Data tiering instances are ideal for workloads that often access a small portion of your overall dataset but still requires fast access to the remaining data. In other words, the ratio of hot to warm data is about 20:80.
  2. Develop cluster level tacking of object idle time.
  3. Large implementations of over 500 Gb of data are good candidates
- **[Required]** Understand that data tiering instances are not optional for certain workloads.

1. There is a small performance cost for accessing less frequently used objects as those are swapped out to local SSD. If your application is response time sensitive test the impact on your workload.
2. Not suitable for caches that store mostly large objects over 128 MiB in size.

**[Resources]:**

- Limitations

- **[Best]** Reserved instance types support data tiering. This assures the lowest cost in terms of amount of data storage per instance.
  1. You may need to operate your ElastiCache clusters using non-data tiering instances until you have a better understanding of your requirements.
  2. Analyze your ElastiCache clusters data usage pattern.
  3. Create an automated job that periodically collects object idle time.
  4. If you notice that a large percentage (about 80%) of objects are idle for a period of time deemed appropriate for your workload document the findings and suggest migrating the cluster to instances that support data tiering.
  5. Regularly evaluate new cache node types available and assess whether it makes sense, from a cost and operational metrics perspective, to migrate your instance fleet to new cache node types.

  **[Resources]:**

  - OBJECT IDLETIME
  - Amazon ElastiCache pricing

# Common troubleshooting steps and best practices with ElastiCache

The following topics provide troubleshooting advice for errors and issues that you could encounter when using ElastiCache. If you find an issue that isn't listed here, you can use the feedback button on this page to report it.

For more troubleshooting advice and answers to common support questions, visit the AWS Knowledge Center

**Topics**

- Connection issues
- Valkey or Redis OSS client errors
- Troubleshooting high latency in ElastiCache Serverless
- Troubleshooting throttling issues in ElastiCache Serverless
- Persistent connection issues
- Related Topics


## Connection issues

If you are unable to connect to your ElastiCache cache, consider one of the following:

1. **Using TLS:** If you are experiencing a hung connection when trying to connect to your ElastiCache endpoint, you may not be using TLS in your client. If you are using ElastiCache Serverless, encryption in transit is always enabled. Make sure that your client is using TLS to connect to the cache. For serverless caches with a public endpoint, your client must support TLS 1.3. Learn more about connecting to a TLS enabled cache.
2. **VPC:** ElastiCache node-based clusters and Serverless caches with a VPC endpoint are accessible from within a VPC. For these caches, ensure that the EC2 instance from which you are accessing the cache and the ElastiCache cache are created in the same VPC. Alternatively, you must enable VPC peering between the VPC where your EC2 instance resides and the VPC where you are creating your cache. Serverless caches with a public endpoint are accessible over the internet and do not require this VPC configuration.
3. **Security groups:** For caches with a VPC endpoint, ElastiCache uses security groups to control access to your cache. Caches with a public endpoint do not use security groups. For VPC caches, consider the following:
   a. Make sure that the security group used by your ElastiCache cache allows inbound access to it from your EC2 instance. See here to learn how to setup inbound rules in your security group correctly.
   b. Make sure that the security group used by your ElastiCache cache allows access to your cache's ports ( 6379 and 6380 for serverless, and 6379 by default for node-based clusters). ElastiCache uses these ports to accept Valkey or Redis OSS commands. Learn more about how to setup port access here.

If connection continues to be difficult, see Persistent connection issues for other steps.

## Valkey or Redis OSS client errors

ElastiCache Serverless is only accessible using clients that support the Valkey or Redis OSS cluster mode protocol. Node-based clusters can be accessed from clients in either mode, depending on the cluster configuration.

If you are experiencing errors in your client, consider the following:

1. **Cluster mode:** If you are experiencing CROSSLOT errors or errors with the SELECT command, you may be trying to access a Cluster Mode Enabled cache with a Valkey or Redis OSS client that does not support the Cluster protocol. ElastiCache Serverless only supports clients that support the Valkey or Redis OSS cluster protocol. If you want to use Valkey or Redis OSS in "Cluster Mode Disabled" (CMD), then you must create a node-based cluster.
2. **CROSSLOT errors:** If you are experiencing the `ERR CROSSLOT Keys in request don't hash to the same slot` error, you may be attempting to access keys that do not belong to the same slot in a Cluster mode cache. As a reminder, ElastiCache Serverless always operates in Cluster Mode. Multi-key operations, transactions, or Lua scripts involving multiple keys are allowed only if all the keys involved are in the same hash slot.

For additional best practices around configuring Valkey or Redis OSS clients, please review this blog post.

## Troubleshooting high latency in ElastiCache Serverless

If your workload appears to experience high latency, you can analyze the CloudWatch `SuccessfulReadRequestLatency` and `SuccessfulWriteRequestLatency` metrics to check if the latency is related to ElastiCache Serverless. These metrics measure latency which is internal to ElastiCache Serverless - client side latency and network trip times between your client and the ElastiCache Serverless endpoint are not included.

### Troubleshooting client-side latency

If you notice elevated latency on the client side but no corresponding increase in CloudWatch `SuccessfulReadRequestLatency` and `SuccessfulWriteRequestLatency` metrics which measure the server-side latency, consider the following:

- **Ensure the security group allows access to ports 6379 and 6380:** ElastiCache Serverless uses the 6379 port for the primary endpoint and the 6380 port for the reader endpoint. Some clients

establish connectivity to both ports for every new connection, even if your application is not using the Read from Replica feature. If your security group does not allow inbound access to both ports, then connection establishment can take longer. Learn more about how to setup port access here.

**Troubleshooting server-side latency**

Some variability and occasional spikes should not be a cause for concern. However, if the `Average` statistic shows a sharp increase and persists, you should check the Health Dashboard and your Personal Health Dashboard for more information. If necessary, consider opening a support case with Support.

Consider the following best practices and strategies to reduce latency:

- **Enable Read from Replica:** If your application allows it, we recommend enabling the "Read from Replica" feature in your Valkey or Redis OSS client to scale reads and achieve lower latency. When enabled, ElastiCache Serverless attempts to route your read requests to replica cache nodes that are in the same Availability Zone (AZ) as your client, thus avoiding cross-AZ network latency. Note, that enabling the Read from Replica feature in your client signifies that your application accepts eventual consistency in data. Your application may receive older data for some time if you attempt to read after writing to a key.
- **Account for changes to the primary node's AZ:** After events such as patching or a failover, ElastiCache Serverless can promote a node in a different AZ to primary. Writes, and reads that are not served from a replica, always go to the primary. If your client runs in a single AZ, these requests were previously local to the primary, and after the change all of them cross to the remote AZ, which raises latency. You see this as a step increase in `SuccessfulReadRequestLatency` and `SuccessfulWriteRequestLatency` that persists until the primary moves again. This latency increase is expected behavior for ElastiCache Serverless, and you can expect it to stay within the normal range.
- **Distribute your client across multiple AZs:** Deploy your client in two or more of your cache's AZs. This routes a portion of your writes and reads served from the primary locally, which reduces average latency. A client spread evenly across two AZs routes about half of these requests locally.
- **Ensure your application is deployed in the same AZs as your cache:** You may observe higher client side latency if your application is not deployed in the same AZs as your cache. When you create a serverless cache you can provide the subnets from where your application will access the cache, and ElastiCache Serverless creates VPC Endpoints in those subnets. Ensure that your

application is deployed in the same AZs. Otherwise, your application may incur a cross-AZ hop when accessing the cache resulting in higher client side latency.

- **Reuse connections:** ElastiCache Serverless requests are made via a TLS enabled TCP connection using the RESP protocol. Initiating the connection (including authenticating the connection, if configured) takes time so the latency of the first request is higher than typical. Requests over an already initialized connection deliver ElastiCache's consistent low latency. For this reason, you should consider using connection pooling or reusing existing Valkey or Redis OSS connections.
- **Scaling speed:** ElastiCache Serverless automatically scales as your request rate grows. A sudden large increase in request rate, faster than the speed at which ElastiCache Serverless scales, may result in elevated latency for some time. ElastiCache Serverless can typically increase its supported request rate quickly, taking up to 10-12 minutes to double the request rate.
- **Inspect long running commands:** Some Valkey or Redis OSS commands, including Lua scripts or commands on large data structures, may run for a long time. To identify these commands, ElastiCache publishes command level metrics. With ElastiCache Serverless you can use the `BasedECPUs` metrics.
- **Throttled Requests:** When requests are throttled in ElastiCache Serverless, you may experience an increase in client side latency in your application. When requests are throttled in ElastiCache Serverless, you should see an increase in the `ThrottledCmds` ElastiCache Serverless metric. Review the section below for troubleshooting throttled requests.
- **Uniform distribution of keys and requests:** In ElastiCache for Valkey and Redis OSS, an uneven distribution of keys or requests per slot can result in a hot slot which can result in elevated latency. ElastiCache Serverless supports up to 30,000 ECPUs/second (90,000 ECPUs/second when using Read from Replica) on a single slot, in a workload that executes simple SET/GET commands. We recommend evaluating your key and request distribution across slots and ensuring a uniform distribution if your request rate exceeds this limit.

## Troubleshooting throttling issues in ElastiCache Serverless

In service-oriented architectures and distributed systems, limiting the rate at which API calls are processed by various service components is called throttling. This smooths spikes, controls for mismatches in component throughput, and allows for more predictable recoveries when there's an unexpected operational event. ElastiCache Serverless is designed for these types of architectures, and most Valkey or Redis OSS clients have retries built in for throttled requests. Some degree of throttling is not necessarily a problem for your application, but persistent throttling of a latency-

sensitive part of your data workflow can negatively impact user experience and reduce the overall efficiency of the system.

When requests are throttled in ElastiCache Serverless, you should see an increase in the `ThrottledCmds` ElastiCache Serverless metric. If you are noticing a high number of throttled requests, consider the following:

- **Scaling speed:** ElastiCache Serverless automatically scales as you ingest more data or grow your request rate.If your application scales faster than the speed at which ElastiCache Serverless scales, then your requests may get throttled while ElastiCache Serverless scales to accommodate your workload. ElastiCache Serverless can typically increase the storage size quickly, taking up to 10-12 minutes to double the storage size in your cache.
- **Uniform distribution of keys and requests:** In ElastiCache for Valkey and Redis OSS, an uneven distribution of keys or requests per slot can result in a hot slot. A hot slot can result in throttling of requests, if the request rate to a single slot exceeds 30,000 ECPUs/second and is in a workload that executes simple SET/GET commands. Similarly, with ElastiCache for Memcached, a hot key can result in throttling of requests if the request rate exceeds 30,000 ECPUs/second.
- **Read from Replica:** If you application allows it, consider using the “Read from Replica“ feature. Most Valkey or Redis OSS clients can be configured to ”scale reads“ to direct reads to replica nodes. This feature enables you to scale read traffic. In addition ElastiCache Serverless automatically routes read from replica requests to nodes in the same Availability Zone as your application resulting in lower latency. When Read from Replica is enabled, you can achieve up to 90,000 ECPUs/second on a single slot, for workloads with simple SET/GET commands.

## Persistent connection issues

The following items must be verified while troubleshooting persistent connectivity issues with ElastiCache:

**Topics**


- Security groups
- Network ACLs
- Route tables
- DNS resolution
- Identifying issues with server-side diagnostics
- Network connectivity validation

- Network-related limits
- CPU Usage
- Connections being terminated from the server side
- Client-side troubleshooting for Amazon EC2 instances
- Dissecting the time taken to complete a single request

## Security groups

Security Groups are virtual firewalls protecting your ElastiCache client (EC2 instance, AWS Lambda function, Amazon ECS container, etc.) and ElastiCache cache. Security groups are stateful, meaning that after the incoming or outgoing traffic is allowed, the responses for that traffic will be automatically authorized in the context of that specific security group.

The stateful feature requires the security group to keep track of all authorized connections, and there is a limit for tracked connections. If the limit is reached, new connections will fail. Please refer to the troubleshooting section for help on how to identify if the limits has been hit on the client or ElastiCache side.

You can have a single security groups assigned at the same time to the client and ElastiCache cluster, or individual security groups for each.

For both cases, you need to allow the TCP outbound traffic on the ElastiCache port from the source and the inbound traffic on the same port to ElastiCache. The default port is 11211 for Memcached, and 6379 for Valkey or Redis OSS. For Amazon ElastiCache Serverless, you must allow both port 6379 and port 6380. Port 6379 is the primary endpoint for read and write operations, and port 6380 is the read-optimized endpoint. Many client libraries connect to both ports even if you don't actively use Read From Replica, so you must make both ports accessible. By default, security groups allow all outbound traffic. In this case, only the inbound rule in the target security group is required.

For more information, see Access patterns for accessing an ElastiCache cluster in an Amazon VPC.

## Network ACLs

Network Access Control Lists (ACLs) are stateless rules. The traffic must be allowed in both directions (Inbound and Outbound) to succeed. Network ACLs are assigned to subnets, not specific resources. It is possible to have the same ACL assigned to ElastiCache and the client resource, especially if they are in the same subnet.

By default, network ACLs allow all trafic. However, it is possible to customize them to deny or allow traffic. Additionally, the evaluation of ACL rules is sequential, meaning that the rule with the lowest number matching the traffic will allow or deny it. The minimum configuration to allow the Valkey or Redis OSS traffic is:

Client side Network ACL:

- **Inbound Rules:**
- Rule number: preferably lower than any deny rule;
- Type: Custom TCP Rule;
- Protocol: TCP
- Port Range: 1024-65535
- Source: 0.0.0.0/0 (or create individial rules for the ElastiCache cluster subnets)
- Allow/Deny: Allow

- **Outbound Rules:**
- Rule number: preferably lower than any deny rule;
- Type: Custom TCP Rule;
- Protocol: TCP
- Port Range: 6379 (or 6379–6380 for Amazon ElastiCache Serverless)
- Source: 0.0.0.0/0 (or the ElastiCache cluster subnets. Keep in mind that using specific IPs may create issues in case of failover or scaling the cluster)
- Allow/Deny: Allow

ElastiCache Network ACL:

- **Inbound Rules:**
- Rule number: preferably lower than any deny rule;
- Type: Custom TCP Rule;
- Protocol: TCP
- Port Range: 6379 (or 6379–6380 for Amazon ElastiCache Serverless)
- Source: 0.0.0.0/0 (or create individual rules for the ElastiCache cluster subnets)
- Allow/Deny: Allow

- **Outbound Rules:**
- Rule number: preferably lower than any deny rule;
- Type: Custom TCP Rule;
- Protocol: TCP
- Port Range: 1024-65535
- Source: 0.0.0.0/0 (or the ElastiCache cluster subnets. Keep in mind that using specific IPs may create issues in case of failover or scaling the cluster)
- Allow/Deny: Allow

For more information, see Network ACLs.

### Route tables

Similarly to Network ACLs, each subnet can have different route tables. If clients and the ElastiCache cluster are in different subnets, make sure that their route tables allow them to reach each other.

More complex environments, involving multiple VPCs, dynamic routing, or network firewalls, may become difficult to troubleshoot. See Network connectivity validation to confirm that your network settings are appropriate.

### DNS resolution

ElastiCache provides the service endpoints based on DNS names. The endpoints available are `Configuration`, `Primary`, `Reader`, and `Node` endpoints. For more information, see Finding Connection Endpoints.

In case of failover or cluster modification, the address associated to the endpoint name may change and will be automatically updated.

Custom DNS settings (i.e., not using the VPC DNS service) may not be aware of the ElastiCache-provided DNS names. Make sure that your system can successfully resolve the ElastiCache endpoints using system tools like `dig` (as shown following) or `nslookup`.

```
$ dig +short example.xxxxxx.ng.0001.use1.cache.amazonaws.com
example-001.xxxxxx.0001.use1.cache.amazonaws.com.
1.2.3.4
```

You can also force the name resolution through the VPC DNS service:

```
$ dig +short example.xxxxxx.ng.0001.use1.cache.amazonaws.com @169.254.169.253
example-001.tihewd.0001.use1.cache.amazonaws.com.
1.2.3.4
```

## Identifying issues with server-side diagnostics

CloudWatch metrics and run-time information from the ElastiCache engine are common sources or information to identify potential sources of connection issues. A good analysis commonly starts with the following items:

- CPU usage: Valkey and Redis OSS are multi-threaded applications. However, execution of each command happens in a single (main) thread. For this reason, ElastiCache provides the metrics `CPUUtilization` and `EngineCPUUtilization`. `EngineCPUUtilization` provides the CPU utilization dedicated to the Valkey or Redis OSS process, and `CPUUtilization` the usage across all vCPUs. Nodes with more than one vCPU usually have different values for `CPUUtilization` and `EngineCPUUtilization`, the second being commonly higher. High `EngineCPUUtilization` can be caused by an elevated number of requests or complex operations that take a significant amount of CPU time to complete. You can identify both with the following:
  - Elevated number of requests: Check for increases on other metrics matching the `EngineCPUUtilization` pattern. Useful metrics are:
    - `CacheHits` and `CacheMisses`: the number of successful requests or requests that didn't find a valid item in the cache. If the ratio of misses compared to hits is high, the application is wasting time and resources with unfruitful requests.
    - `SetTypeCmds` and `GetTypeCmds`: These metrics correlated with `EngineCPUUtilization` can help to understand if the load is significantly higher for write requests, measured by `SetTypeCmds`, or reads, measured by `GetTypeCmds`. If the load is predominantly reads, using multiple read-replicas can balance the requests across multiple nodes and spare the primary for writes. In cluster mode-disabled clusters, the use of read-replicas can be done by creating an additional connection configuration in the application using the ElastiCache reader endpoint. For more information, see Finding Connection Endpoints. The read operations must be submitted to this additional connection. Write operations will be done through the regular primary endpoint. In cluster mode-enabled, it is advisable to use a library that supports read replicas natively. With the right flags, the library will be able to automatically discover the cluster topology, the replica nodes, enable the read operations

through the READONLY Valkey or Redis OSS command, and submit the read requests to the replicas.

- Elevated number of connections:
  - `CurrConnections` and `NewConnections`: `CurrConnection` is the number of established connections at the moment of the datapoint collection, while `NewConnections` shows how many connections were created in the period.

    Creating and handling connections implies significant CPU overhead. Additionally, the TCP three-way handshake required to create new connections will negatively affect the overall response times.

    An ElastiCache node with thousands of `NewConnections` per minute indicates that a connection is created and used by just a few commands, which is not optimal. Keeping connections established and reusing them for new operations is a best practice. This is possible when the client application supports and properly implements connection pooling or persistent connections. With connection pooling, the number of `currConnections` does not have big variations, and the `NewConnections` should be as low as possible. Valkey and Redis OSS provide optimal performance with small number of currConnections. Keeping currConnection in the order of tens or hundreds minimizes the usage of resources to support individual connections like client buffers and CPU cycles to serve the connection.
- Network throughput:
  - Determine the bandwidth: ElastiCache nodes have network bandwidth proportional to the node size. Since applications have different characteristics, the results can vary according to the workload. As examples, applications with high rate of small requests tend to affect more the CPU usage than the network throughput while bigger keys will cause higher network utilization. For that reason, it is advisable to test the nodes with the actual workload for a better understanding of the limits.

    Simulating the load from the application would provide more accurate results. However, benchmark tools can give a good idea of the limits.
  - For cases where the requests are predominantly reads, using replicas for read operations will alleviate the load on the primary node. If the use-case is predominantly writes, the use of many replicas will amplify the network usage. For every byte written to the primary node, N bytes will be sent to the replicas,being N the number of replicas. The best practice for write intensive workloads are using ElastiCache for Redis OSS with cluster mode-enabled so the

writes can be balanced across multiple shards, or scale-up to a node type with more network capabilities.

- The CloudWatchmetrics `NetworkBytesIn` and `NetworkBytesOut` provide the amount of data coming into or leaving the node, respectively. `ReplicationBytes` is the traffic dedicated to data replication.

For more information, see Network-related limits.

- Complex commands: Redis OSS commands are served on a single thread, meaning that requests are served sequentially. A single slow command can affect other requests and connections, culminating in time-outs. The use of commands that act upon multiple values, keys, or data types must be done carefully. Connections can be blocked or terminated depending on the number of parameters, or size of its input or output values.

  A notorious example is the `KEYS` command. It sweeps the entire keyspace searching for a given pattern and blocks the execution of other commands during its execution. Redis OSS uses the "Big O" notation to describe its commands complexity.

  Keys command has O(N) time complexity, N being the number of keys in the database. Therefore, the larger the number of keys, the slower the command will be. `KEYS` can cause trouble in different ways: If no search pattern is used, the command will return all key names available. In databases with thousand or million of items, a huge output will be created and flood the network buffers.

  If a search pattern is used, only the keys matching the pattern will return to the client. However, the engine still sweeps the entire keyspace searching for it, and the time to complete the command will be the same.

  An alternative for `KEYS` is the `SCAN` command. It iterates over the keyspace and limits the iterations in a specific number of items, avoiding prolonged blocks on the engine.

  Scan has the `COUNT` parameter, used to set the size of the iteration blocks. The default value is 10 (10 items per iteration).

  Depending on the number of items in the database, small `COUNT` values blocks will require more iterations to complete a full scan, and bigger values will keep the engine busy for longer at each iteration. While small count values will make `SCAN` slower on big databases, larger values can cause the same issues mentioned for `KEYS`.

As an example, running the SCAN command with count value as 10 will requires 100,000 repetitions on a database with 1 million keys. If the average network round-trip time is 0.5 milliseconds, approximately 50,000 milliseconds (50 seconds) will be spent transferring requests.

On the other hand, if the count value were 100,0000, a single iteration would be required and only 0.5 ms would be spent transferring it. However, the engine would be entirely blocked for other operations until the command finishes sweeping all the keyspace.

Besides KEYS, several other commands are potentially harmful if not used correctly. To see a list of all commands and their respective time complexity, go to Valkey and Redis OSS commands.

Examples of potential issues:

- Lua scripts: Valkey and Redis OSS provide an embedded Lua interpreter, allowing the execution of scripts on the server-side. Lua scripts on Valkey and Redis OSS are executed on engine level and are atomic by definition, meaning that no other command or script will be allowed to run while a script is in execution. Lua scripts provide the possibility of running multiple commands, decision-making algorithms, data parsing, and others directly on the engine. While the atomicity of scripts and the chance of offloading the application are tempting, scripts must be used with care and for small operations. On ElastiCache, the execution time of Lua scripts is limited to 5 seconds. Scripts that haven't written to the keyspace will be automatically terminated after the 5 seconds period. To avoid data corruption and inconsistencies, the node will failover if the script execution hasn't completed in 5 seconds and had any write during its execution. Transactions are the alternative to guarantee consistency of multiple related key modifications in Redis OSS. A transaction allows the execution of a block of commands, watching existing keys for modifications. If any of the watched keys changes before the completion of the transaction, all modifications are discarded.
- Mass deletion of items: The DEL command accepts multiple parameters, which are the key names to be deleted. Deletion operations are synchronous and will take significant CPU time if the list of parameters is big, or contains a big list, set, sorted set, or hash (data structures holding several sub-items). In other words, even the deletion of a single key can take significant time if it has many elements. The alternative to DEL is UNLINK, which is an asynchronous command available since Redis OSS 4. UNLINK must be preferred over DEL whenever possible. Starting on ElastiCache for Redis OSS 6x, the `lazyfree-lazy-`

`user-del` parameter makes the `DEL` command behave like `UNLINK` when enabled. For more information, see Redis OSS 6.0 Parameter Changes.

- Commands acting upon multiple keys: `DEL` was mentioned before as a command that accepts multiple arguments and its execution time will be directly proportional to that. However, Redis OSS provides many more commands that work similarly. As examples, `MSET` and `MGET` allow the insertion or retrieval of multiple String keys at once. Their usage may be beneficial to reduce the network latency inherent to multiple individual `SET` or `GET` commands. However, an extensive list of parameters will affect CPU usage.

  While CPU utilization alone is not the cause for connectivity issues, spending too much time to process a single or few commands over multiple keys may cause failure of other requests and increase overall CPU utilization.

  The number of keys and their size will affect the command complexity and consequently completion time.

  Other examples of commands that can act upon multiple keys: `HMGET`, `HMSET`, `MSETNX`, `PFCOUNT`, `PFMERGE`, `SDIFF`, `SDIFFSTORE`, `SINTER`, `SINTERSTORE`, `SUNION`, `SUNIONSTORE`, `TOUCH`, `ZDIFF`, `ZDIFFSTORE`, `ZINTER` or `ZINTERSTORE`.
- Commands acting upon multiple data types: Redis OSS also provides commands that act upon one or multiple keys, regardless of their data type. ElastiCache for Redis OSS provides the metric `KeyBasedCmds` to monitor such commands. This metric sums the execution of the following commands in the selected period:
  - O(N) complexity:
    - `KEYS`
  - O(1)
    - `EXISTS`
    - `OBJECT`
    - `PTTL`
    - `RANDOMKEY`
    - `TTL`
    - `TYPE`
    - `EXPIRE`
    - `EXPIREAT`

    - MOVE
    - PERSIST
    - PEXPIRE
    - PEXPIREAT
    - UNLINK (O(N) to reclaim memory. However the memory-reclaiming task happens in a separated thread and does not block the engine
  - Different complexity times depending on the data type:
    - DEL
    - DUMP
    - RENAME is considered a command with O(1) complexity, but executes DEL internally. The execution time will vary depending on the size of the renamed key.
    - RENAMENX
    - RESTORE
    - SORT
  - Big hashes: Hash is a data type that allows a single key with multiple key-value sub-items. Each hash can store 4.294.967.295 items and operations on big hashes can become expensive. Similarly to KEYS, hashes have the HKEYS command with O(N) time complexity, N being the number of items in the hash. HSCAN must be preferred over HKEYS to avoid long running commands. HDEL, HGETALL, HMGET, HMSET and HVALS are commands that should be used with caution on big hashes.
- Other big data-structures: Besides hashes, other data structures can be CPU intensive. Sets, Lists, Sorted Sets, and Hyperloglogs can also take significant time to be manipulated depending on their size and commands used. For more information on those commands, see Valkey and Redis OSS commands.

## Network connectivity validation

After reviewing the network configurations related to DNS resolution, security groups, network ACLs, and route tables, the connectivity can be validated with the VPC Reachability Analyzer and system tools.

Reachability Analyzer will test the network connectivity and confirm if all the requirements and permissions are satisfied. For the tests below you will need the ENI ID (Elastic Network Interface

Identification) of one of the ElastiCache nodes available in your VPC. You can find it by doing the following:

1. Go to https://console.aws.amazon.com/ec2/v2/home?#NIC:
2. Filter the interface list by your ElastiCache cluster name or the IP address got from the DNS validations previously.
3. Write down or otherwise save the ENI ID. If multiple interfaces are shown, review the description to confirm that they belong to the right ElastiCache cluster and choose one of them.
4. Proceed to the next step.
5. Create an analyze path at https://console.aws.amazon.com/vpc/home?#ReachabilityAnalyzer and choose the following options:

   - **Source Type**: Choose **instance** if your ElastiCache client runs on an Amazon EC2 instance or **Network Interface** if it uses another service, such as AWS Fargate Amazon ECS with awsvpc network, AWS Lambda, etc), and the respective resource ID (EC2 instance or ENI ID);
   - **Destination Type**: Choose **Network Interface** and select the **Elasticache ENI** from the list.
   - **Destination port**: specify 6379 for ElastiCache for Redis OSS or 11211 for ElastiCache for Memcached. Those are the ports defined with the default configuration and this example assumes that they are not changed.
   - **Protocol**: TCP

Create the analyze path and wait a few moments for the result. If the status is unreachable, open the analysis details and review the **Analysis explorer** for details where the requests were blocked.

If the reachability tests passed, proceed to the verification on the OS level.

To validate the TCP connectivity on the ElastiCache service port: On Amazon Linux, `Nping` is available in the package `nmap` and can test the TCP connectivity on the ElastiCache port, as well as providing the network round-trip time to establish the connection. Use this to validate the network connectivity and the current latency to the ElastiCache cluster, as shown following:

```
$ sudo nping --tcp -p 6379 example.xxxxxx.ng.0001.use1.cache.amazonaws.com

Starting Nping 0.6.40 ( http://nmap.org/nping ) at 2020-12-30 16:48 UTC
SENT (0.0495s) TCP ...
(Output suppressed )
```

```
Max rtt: 0.937ms | Min rtt: 0.318ms | Avg rtt: 0.449ms
Raw packets sent: 5 (200B) | Rcvd: 5 (220B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 4.08 seconds
```

By default, `nping` sends 5 probes with a delay of 1 second between them. You can use the option "-c" to increase the number of probes and "--delay" to change the time to send a new test.

If the tests with `nping` fail and the *VPC Reachability Analyzer* tests passed, ask your system administrator to review possible Host-based firewall rules, asymmetric routing rules, or any other possible restriction on the operating system level.

On the ElastiCache console, check if **Encryption in-transit** is enabled in your ElastiCache cluster details. If in-transit encryption is enabled, confirm if the TLS session can be established with the following command:

```
openssl s_client -connect example.xxxxxx.use1.cache.amazonaws.com:6379
```

An extensive output is expected if the connection and TLS negotiation are successful. Check the return code available in the last line, the value must be `0 (ok)`. If openssl returns something different, check the reason for the error at https://www.openssl.org/docs/man1.0.2/man1/verify.html#DIAGNOSTICS.

If all the infrastructure and operating system tests passed but your application is still unable to connect to ElastiCache, check if the application configurations are compliant with the ElastiCache settings. Common mistakes are:

- Your application does not support ElastiCache cluster mode, and ElastiCache has cluster-mode enabled;
- Your application does not support TLS/SSL, and ElastiCache has in-transit encryption enabled;
- Application supports TLS/SSL but does not have the right configuration flags or trusted certification authorities;

## Network-related limits

- Maximum number of connections: There are hard limits for simultaneous connections. Each ElastiCache node allows up to 65,000 simultaneous connections across all clients. This limit can be monitored through the `CurrConnections` metrics on CloudWatch. However, clients also

have their limits for outbound connections.On Linux, check the allowed ephemeral port range with the command:

```
# sysctl net.ipv4.ip_local_port_range
net.ipv4.ip_local_port_range = 32768 60999
```

In the previous example, 28231 connections will be allowed from the same source, to the same destination IP (ElastiCache node) and port. The following command shows how many connections exist for a specific ElastiCache node (IP 1.2.3.4):

```
ss --numeric --tcp state connected "dst 1.2.3.4 and dport == 6379" | grep -vE
 '^State' | wc -l
```

If the number is too high, your system may become overloaded trying to process the connection requests. It is advisable to consider implementing techniques like connection pooling or persistent connections to better handle the connections. Whenever possible, configure the connection pool to limit the maximum number of connections to a few hundred. Also, back-off logic to handle time-outs or other connection exceptions would are advisable to avoid connection churn in case of issues.

- Network traffic limits: Check the following CloudWatch metrics for Redis OSS to identify possible network limits being hit on the ElastiCache node:
  - `NetworkBandwidthInAllowanceExceeded` / `NetworkBandwidthOutAllowanceExceeded`: Network packets queued or dropped because the throughput exceeded the aggregated bandwidth limit.

    It is important to note that every byte written to the primary node will be replicated to N replicas, N being the number of replicas. Clusters with small node types, multiple replicas, and intensive write requests may not be able to cope with the replication backlog. For such cases, it's a best practice to scale-up (change node type), scale-out (add shards in cluster-mode enabled clusters), reduce the number of replicas, or minimize the number of writes.
  - `NetworkConntrackAllowanceExceeded`: Packets queued or dropped because the maximum number of connections tracked across all security groups assigned to the node has been exceeded. New connections will likely fail during this period.
  - `NetworkPackets PerSecondAllowanceExceeded`: Maximum number of packets per second exceeded. Workloads based on a high rate of very small requests may hit this limit before the maximum bandwidth.

The metrics above are the ideal way to confirm nodes hitting their network limits. However, limits are also identifiable by plateaus on network metrics.

If the plateaus are observed for extended periods, they will be likely followed by replication lag, increase on bytes Used for cache, drop on freeable memory, high swap and CPU usage. Amazon EC2 instances also have network limits that can tracked through ENA driver metrics. Linux instances with enhanced networking support and ENA drivers 2.2.10 or newer can review the limit counters with the command:

```
# ethtool -S eth0 | grep "allowance_exceeded"
```

## CPU Usage

The CPU usage metric is the starting point of investigation, and the following items can help to narrow down possible issues on the ElastiCache side:

- Redis OSS SlowLogs: The ElastiCache default configuration retains the last 128 commands that took over 10 milliseconds to complete. The history of slow commands is kept during the engine runtime and will be lost in case of failure or restart. If the list reaches 128 entries, old events will be removed to open room for new ones. The size of the list of slow events and the execution time considered slow can by modified via the parameters `slowlog-max-len` and `slowlog-log-slower-than` in a custom parameter group. The slowlogs list can be retrieved by running `SLOWLOG GET 128` on the engine, 128 being the last 128 slow commands reported. Each entry has the following fields:

```
1) 1) (integer) 1 -----------> Sequential ID
   2) (integer) 1609010767 --> Timestamp (Unix epoch time)of the Event
   3) (integer) 4823378 -----> Time in microseconds to complete the command.
   4) 1) "keys" -------------> Command
      2) "*" ----------------> Arguments
   5) "1.2.3.4:57004"-> Source
```

The event above happened on December 26, at 19:26:07 UTC, took 4.8 seconds (4.823ms) to complete and was caused by the KEYS command requested from the client 1.2.3.4.

On Linux, the timestamp can be converted with the command date:

```
$ date --date='@1609010767'
Sat Dec 26 19:26:07 UTC 2020
```

With Python:

```
>>> from datetime import datetime
>>> datetime.fromtimestamp(1609010767)
datetime.datetime(2020, 12, 26, 19, 26, 7)
```

Or on Windows with PowerShell:

```
PS D:\Users\user> [datetimeoffset]::FromUnixTimeSeconds('1609010767')
DateTime      : 12/26/2020 7:26:07 PM
UtcDateTime
                : 12/26/2020 7:26:07 PM
LocalDateTime : 12/26/2020 2:26:07 PM
Date          : 12/26/2020 12:00:00 AM
Day           : 26
DayOfWeek
                : Saturday
DayOfYear     : 361
Hour          : 19
Millisecond   : 0
Minute        : 26
Month
                : 12
Offset        : 00:00:00Ticks         : 637446075670000000
UtcTicks
                : 637446075670000000
TimeOfDay     : 19:26:07
Year          : 2020
```

Many slow commands in a short period of time (same minute or less) is a reason for concern. Review the nature of commands and how they can be optimized (see previous examples). If commands with O(1) time complexity are frequently reported, check the other factors for high CPU usage mentioned before.

- Latency metrics: ElastiCache for Redis OSS provides CloudWatch metrics to monitor the average latency for different classes of commands. The datapoint is calculated by dividing the total number of executions of commands in the category by the total execution time in the period. It

is important to understand that latency metric results are an aggregate of multiple commands. A single command can cause unexpected results, like timeouts, without significant impact on the metrics. For such cases, the slowlog events would be a more accurate source of information. The following list contains the latency metrics available and the respective commands that affect them.

- EvalBasedCmdsLatency: related to Lua Script commands, `eval`, `evalsha`;
- GeoSpatialBasedCmdsLatency: `geodist`, `geohash`, `geopos`, `georadius`, `georadiusbymember`, `geoadd`;
- GetTypeCmdsLatency: Read commands, regardless of data type;
- HashBasedCmdsLatency: `hexists`, `hget`, `hgetall`, `hkeys`, `hlen`, `hmget`, `hvals`, `hstrlen`, `hdel`, `hincrby`, `hincrbyfloat`, `hmset`, `hset`, `hsetnx`;
- HyperLogLogBasedCmdsLatency: `pfselftest`, `pfcount`, `pfdebug`, `pfadd`, `pfmerge`;
- KeyBasedCmdsLatency: Commands that can act upon different data types: `dump`, `exists`, `keys`, `object`, `pttl`, `randomkey`, `ttl`, `type`, `del`, `expire`, `expireat`, `move`, `persist`, `pexpire`, `pexpireat`, `rename`, `renamenx`, `restoreK`, `sort`, `unlink`;
- ListBasedCmdsLatency: lindex, llen, lrange, blpop, brpop, brpoplpush, linsert, lpop, lpush, lpushx, lrem, lset, ltrim, rpop, rpoplpush, rpush, rpushx;
- PubSubBasedCmdsLatency: psubscribe, publish, pubsub, punsubscribe, subscribe, unsubscribe;
- SetBasedCmdsLatency: `scard`, `sdiff`, `sinter`, `sismember`, `smembers`, `srandmember`, `sunion`, `sadd`, `sdiffstore`, `sinterstore`, `smove`, `spop`, `srem`, `sunionstore`;
- SetTypeCmdsLatency: Write commands, regardless of data-type;
- SortedSetBasedCmdsLatency: `zcard`, `zcount`, `zrange`, `zrangebyscore`, `zrank`, `zrevrange`, `zrevrangebyscore`, `zrevrank`, `zscore`, `zrangebylex`, `zrevrangebylex`, `zlexcount`, `zadd`. `zincrby`, `zinterstore`, `zrem`, `zremrangebyrank`, `zremrangebyscore`, `zunionstore`, `zremrangebylex`, `zpopmax`, `zpopmin`, `bzpopmin`, `bzpopmax`;
- StringBasedCmdsLatency: `bitcount`, `get`, `getbit`, `getrange`, `mget`, `strlen`, `substr`, `bitpos`, `append`, `bitop`, `bitfield`, `decr`, `decrby`, `getset`, `incr`, `incrby`, `incrbyfloat`, `mset`, `msetnx`, `psetex`, `set`, `setbit`, `setex`, `setnx`, `setrange`;
- StreamBasedCmdsLatency: `xrange`, `xrevrange`, `xlen`, `xread`, `xpending`, `xinfo`, `xadd`, `xgroup`, `readgroup`, `xack`, `xclaim`, `xdel`, `xtrim`, `xsetid`;

- Redis OSS runtime commands:

- info commandstats: Provides a list of commands executed since the engine started, their cumulative executions number, total execution time, and average execution time per command;
- client list: Provides a list of currently connected clients and relevant information like buffers usage, last command executed, etc;

- Backup and replication: ElastiCache uses a forkless backup and replication method, which may delay writes in order to prevent failures.

  This can cause periods of higher CPU utilization, lead to higher response times and consequently lead to client timeouts during its execution. Always check if the client failures happen during the backup window or the `SaveInProgress` metric was 1 in the period. It is advisable to schedule the backup window for periods of low utilization to minimize the possibility of issues with clients or backup failures.

## Connections being terminated from the server side

The default ElastiCache for Redis OSS configuration keeps the client connections established indefinitely. However, in some cases connection termination may be desirable. For example:

- Bugs in the client application may cause connections to be forgotten and kept established with an idle state. This is called "connection leak“ and the consequence is a steady increase on the number of established connections observed on the `CurrConnections` metric. This behavior can result in saturation on the client or ElastiCache side. When an immediate fix is not possible from the client side, some administrators set a ”timeout“ value in their ElastiCache parameter group. The timeout is the time in seconds allowed for idle connections to persist. If the client doesn’t submit any request in the period, the engine will terminate the connection as soon as the connection reaches the timeout value. Small timeout values may result in unnecessary disconnections and clients will need handle them properly and reconnect, causing delays.
- The memory used to store keys is shared with client buffers. Slow clients with big requests or responses may demand a significant amount of memory to handle its buffers. The default ElastiCache for Redis OSS configurations does not restrict the size of regular client output buffers. If the `maxmemory` limit is hit, the engine will try to evict items to fulfill the buffer usage. In extreme low memory conditions, ElastiCache for Redis OSS might choose to disconnect clients that consume large client output buffers in order to free memory and retain the cluster’s health.

It is possible to limit the size of client buffers with custom configurations and clients hitting the limit will be disconnected. However, clients should be able to handle unexpected disconnections. The parameters to handle buffers size for regular clients are the following:

- client-query-buffer-limit: Maximum size of a single input request;
- client-output-buffer-limit-normal-soft-limit: Soft limit for client connections. The connection will be terminated if stays above the soft limit for more than the time in seconds defined on client-output-buffer-limit-normal-soft-seconds or if it hits the hard limit;
- client-output-buffer-limit-normal-soft-seconds: Time allowed for the connections exceeding the client-output-buffer-limit-normal-soft-limit;
- client-output-buffer-limit-normal-hard-limit: A connection hitting this limit will be immediatelly terminated.

Besides the regular client buffers, the following options control the buffer for replica nodes and Pub/Sub (Publish/Subscribe) clients:

- client-output-buffer-limit-replica-hard-limit;
- client-output-buffer-limit-replica-soft-seconds;
- client-output-buffer-limit-replica-hard-limit;
- client-output-buffer-limit-pubsub-soft-limit;
- client-output-buffer-limit-pubsub-soft-seconds;
- client-output-buffer-limit-pubsub-hard-limit;

## Client-side troubleshooting for Amazon EC2 instances

The load and responsiveness on the client side can also affect the requests to ElastiCache. EC2 instance and operating system limits need to be carefully reviewed while troubleshooting intermittent connectivity or timeout issues. Some key points to observe:

- CPU:
  - EC2 instance CPU usage: Make sure the CPU hasn't been saturated or near to 100 percent. Historical analysis can be done via CloudWatch, however keep in mind that data points granularity is either 1 minute (with detailed monitoring enabled) or 5 minutes;
  - If using burstable EC2 instances, make sure that their CPU credit balance hasn't been depleted. This information is available on the `CPUCreditBalance` CloudWatch metric.

- Short periods of high CPU usage can cause timeouts without reflecting on 100 percent utilization on CloudWatch. Such cases require real-time monitoring with operating-system tools like `top`, `ps` and `mpstat`.

- Network
  - Check if the Network throughput is under acceptable values according to the instance capabilities. For more information, see Amazon EC2 Instance Types
  - On instances with the `ena` Enhanced Network driver, check the ena statistics for timeouts or exceeded limits. The following statistics are useful to confirm network limits saturation:
    - `bw_in_allowance_exceeded` / `bw_out_allowance_exceeded`: number of packets queued or dropped due to excessive inbound or outbound throughput;
    - `conntrack_allowance_exceeded`: number of packets dropped due to security groups connection tracking limits. New connections will fail when this limit is saturated;
    - `linklocal_allowance_exceeded`: number of packets dropped due to excessive requests to instance meta-data, NTP via VPC DNS. The limit is 1024 packets per second for all the services;
    - `pps_allowance_exceeded`: number of packets dropped due to excessive packets per second ratio. The PPS limit can be hit when the network traffic consists on thousands or millions of very small requests per second. ElastiCache traffic can be optimized to make better use of network packets via pipelines or commands that do multiple operations at once like `MGET` instead of `GET`.

## Dissecting the time taken to complete a single request

- On the network: `Tcpdump` and `Wireshark` (tshark on the command line) are handy tools to understand how much time the request took to travel the network, hit the ElastiCache engine and get a return. The following example highlights a single request created with the following command:

```
$ echo ping | nc example.xxxxxx.ng.0001.use1.cache.amazonaws.com 6379
+PONG
```

  In parallel to the command above, tcpdump was in execution and returned:

```
$ sudo tcpdump -i any -nn port 6379 -tt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

```
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
1609428918.917869 IP 172.31.11.142.40966
    > 172.31.11.247.6379: Flags [S], seq 177032944, win 26883, options [mss
 8961,sackOK,TS val 27819440 ecr 0,nop,wscale 7], length 0
1609428918.918071 IP 172.31.11.247.6379 > 172.31.11.142.40966: Flags [S.], seq
 53962565, ack 177032945, win
    28960, options [mss 1460,sackOK,TS val 3788576332 ecr 27819440,nop,wscale 7],
 length 0
1609428918.918091 IP 172.31.11.142.40966 > 172.31.11.247.6379: Flags [.], ack 1, win
 211, options [nop,nop,TS val 27819440 ecr 3788576332], length 0
1609428918.918122
    IP 172.31.11.142.40966 > 172.31.11.247.6379: Flags [P.], seq 1:6, ack 1, win 211,
 options [nop,nop,TS val 27819440 ecr 3788576332], length 5: RESP "ping"
1609428918.918132 IP 172.31.11.142.40966 > 172.31.11.247.6379: Flags [F.], seq 6, ack
    1, win 211, options [nop,nop,TS val 27819440 ecr 3788576332], length 0
1609428918.918240 IP 172.31.11.247.6379 > 172.31.11.142.40966: Flags [.], ack 6, win
 227, options [nop,nop,TS val 3788576332 ecr 27819440], length 0
1609428918.918295
    IP 172.31.11.247.6379 > 172.31.11.142.40966: Flags [P.], seq 1:8, ack 7, win 227,
 options [nop,nop,TS val 3788576332 ecr 27819440], length 7: RESP "PONG"
1609428918.918300 IP 172.31.11.142.40966 > 172.31.11.247.6379: Flags [.], ack 8, win
    211, options [nop,nop,TS val 27819441 ecr 3788576332], length 0
1609428918.918302 IP 172.31.11.247.6379 > 172.31.11.142.40966: Flags [F.], seq 8, ack
 7, win 227, options [nop,nop,TS val 3788576332 ecr 27819440], length 0
1609428918.918307
    IP 172.31.11.142.40966 > 172.31.11.247.6379: Flags [.], ack 9, win 211, options
 [nop,nop,TS val 27819441 ecr 3788576332], length 0
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

From the output above we can confirm that the TCP three-way handshake was completed in 222 microseconds (918091 - 917869) and the ping command was submitted and returned in 173 microseconds (918295 - 918122).

It took 438 microseconds (918307 - 917869) from requesting to closing the connection. Those results would confirm that network and engine response times are good and the investigation can focus on other components.

- On the operating system: `Strace` can help identifying time gaps on the OS level. The analysis of actual applications would be way more extensive and specialized application profilers or debuggers are advisable. The following example just shows if the base operating system

components are working as expected, otherwise further investigation may be required. Using the same Redis OSS `PING` command with `strace` we get:

```
$ echo ping | strace -f -tttt -r -e trace=execve,socket,open,recvfrom,sendto
 nc example.xxxxxx.ng.0001.use1.cache.amazonaws.com (http://
example.xxxxxx.ng.0001.use1.cache.amazonaws.com/)
    6379
1609430221.697712 (+ 0.000000) execve("/usr/bin/nc", ["nc",
 "example.xxxxxx.ng.0001.use"..., "6379"], 0x7fffede7cc38 /* 22 vars */) = 0
1609430221.708955 (+ 0.011231) socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|
SOCK_NONBLOCK, 0) = 3
1609430221.709084
    (+ 0.000124) socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
1609430221.709258 (+ 0.000173) open("/etc/nsswitch.conf", O_RDONLY|O_CLOEXEC) = 3
1609430221.709637 (+ 0.000378) open("/etc/host.conf", O_RDONLY|O_CLOEXEC) = 3
1609430221.709923
    (+ 0.000286) open("/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 3
1609430221.711365 (+ 0.001443) open("/etc/hosts", O_RDONLY|O_CLOEXEC) = 3
1609430221.713293 (+ 0.001928) socket(AF_INET, SOCK_DGRAM|SOCK_CLOEXEC|SOCK_NONBLOCK,
 IPPROTO_IP) = 3
1609430221.717419
    (+ 0.004126) recvfrom(3, "\362|
\201\200\0\1\0\2\0\0\0\0\rnotls20201224\6tihew"..., 2048, 0, {sa_family=AF_INET,
 sin_port=htons(53), sin_addr=inet_addr("172.31.0.2")}, [28->16]) = 155
1609430221.717890 (+ 0.000469) recvfrom(3,
 "\204\207\201\200\0\1\0\1\0\0\0\0\rnotls20201224\6tihew"...,
    65536, 0, {sa_family=AF_INET, sin_port=htons(53),
 sin_addr=inet_addr("172.31.0.2")}, [28->16]) = 139
1609430221.745659 (+ 0.027772) socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
1609430221.747548 (+ 0.001887) recvfrom(0, 0x7ffcf2f2ca50, 8192,
    0, 0x7ffcf2f2c9d0, [128]) = -1 ENOTSOCK (Socket operation on non-socket)
1609430221.747858 (+ 0.000308) sendto(3, "ping\n", 5, 0, NULL, 0) = 5
1609430221.748048 (+ 0.000188) recvfrom(0, 0x7ffcf2f2ca50, 8192, 0, 0x7ffcf2f2c9d0,
 [128]) = -1 ENOTSOCK
    (Socket operation on non-socket)
1609430221.748330 (+ 0.000282) recvfrom(3, "+PONG\r\n", 8192, 0, 0x7ffcf2f2c9d0,
 [128->0]) = 7
+PONG
1609430221.748543 (+ 0.000213) recvfrom(3, "", 8192, 0, 0x7ffcf2f2c9d0, [128->0]) = 0
1609430221.752110
    (+ 0.003569) +++ exited with 0 +++
```

In the example above, the command took a little more than 54 milliseconds to complete (752110 - 697712 = 54398 microseconds).

A significant amount of time, approximately 20ms, was taken to instantiate nc and do the name resolution (from 697712 to 717890), after that, 2ms were required to create the TCP socket (745659 to 747858), and 0.4 ms (747858 to 748330) to submit and receive the response for the request.

## Related Topics

- the section called “Best practices and caching strategies”

# Security in Amazon ElastiCache

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is 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 Amazon ElastiCache, 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 Amazon ElastiCache. The following topics show you how to configure Amazon ElastiCache to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon ElastiCache resources.

**Topics**


- Data protection in Amazon ElastiCache
- Internetwork traffic privacy
- Identity and Access Management for Amazon ElastiCache
- Compliance validation for Amazon ElastiCache
- Resilience in Amazon ElastiCache
- Infrastructure security in AWS ElastiCache
- Service updates in ElastiCache
- Common Vulnerabilities and Exposures (CVE): Security vulnerabilities addressed in ElastiCache

# Data protection in Amazon ElastiCache

The AWS shared responsibility model applies to data protection in AWS ElastiCache (ElastiCache). 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. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data privacy FAQ.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual accounts with 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 TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- 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 personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with ElastiCache or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into ElastiCache or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

When you enable in-transit encryption on a cluster, ElastiCache provisions a TLS certificate for the cluster's endpoint that includes the cluster name. AWS Certificate Manager records publicly trusted TLS certificates in public Certificate Transparency logs, which are append-only and can't have entries removed. As a result, the names of clusters that have in-transit encryption enabled appear in public Certificate Transparency logs. Don't include confidential or sensitive information in cluster names.

**Topics**

- Data security in Amazon ElastiCache

# Data security in Amazon ElastiCache

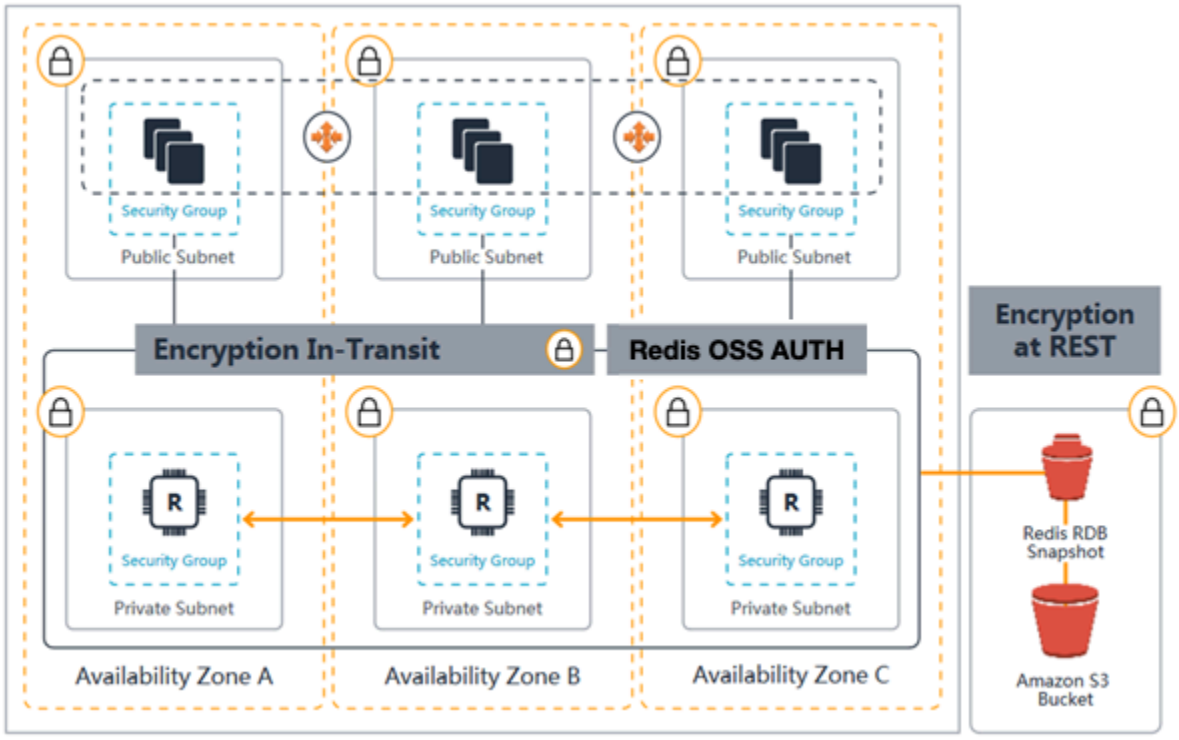
To help keep your data secure, Amazon ElastiCache and Amazon EC2 provide mechanisms to guard against unauthorized access of your data on the server.

Amazon ElastiCache for Memcached provides encryption features for data on caches running Memcached versions 1.6.12 or later.

Amazon ElastiCache with Valkey and Redis OSS provides encryption features for data on caches running Valkey 7.2 or later, and Redis OSS version 4.0.10 or later. Amazon ElastiCache also supports authenticating users with either IAM or Valkey and Redis OSS AUTH, and authorizing user operations using Role-Based Access Control (RBAC).

- In-transit encryption encrypts your data whenever it is moving from one place to another, such as between nodes in your cluster or between your cache and your application.
- At-rest encryption encrypts your on-disk data during sync and backup operations.

ElastiCache supports authenticating users using IAM and the Valkey and Redis OSS AUTH command, and authorizing user operations using Role-Based Access Control (RBAC).




*ElastiCache for Valkey and Redis OSS Security Diagram*

**Topics**


- ElastiCache in-transit encryption (TLS)
- At-Rest Encryption in ElastiCache
- Authentication and Authorization


## ElastiCache in-transit encryption (TLS)

To help keep your data secure, Amazon ElastiCache and Amazon EC2 provide mechanisms to guard against unauthorized access of your data on the server. By providing in-transit encryption capability, ElastiCache gives you a tool you can use to help protect your data when it is moving from one location to another.

All Valkey or Redis OSS serverless caches have in-transit encryption enabled. For node-based clusters, you can enable in-transit encryption on a replication group by setting the parameter `TransitEncryptionEnabled` to `true` (CLI: `--transit-encryption-enabled`) when you create the replication group. You can do this whether you are creating the replication group using the AWS Management Console, the AWS CLI, or the ElastiCache API.

All serverless caches have in-transit encryption enabled. For node-based clusters, you can enable in-transit encryption on a cluster by setting the parameter `TransitEncryptionEnabled` to `true` (CLI: `--transit-encryption-enabled`) when you create the cluster using the `CreateCacheCluster` (CLI: `create-cache-cluster`) operation.

**Topics**


- In-transit encryption overview
- In-transit encryption conditions (Valkey and Redis OSS)
- In-transit encryption conditions (Memcached)
- In-transit encryption best practices
- Further Valkey and Redis OSS options
- Enabling in-transit encryption for Memcached
- Enabling in-transit encryption
- Connecting to ElastiCache (Valkey) or Amazon ElastiCache for Redis OSS with in-transit encryption using valkey-cli
- Enabling in-transit encryption on a node-based Redis OSS cluster using Python

- Best practices when enabling in-transit encryption
- Connecting to nodes enabled with in-transit encryption using Openssl (Memcached)
- Creating a TLS Memcached client using Java
- Creating a TLS Memcached client using PHP

### In-transit encryption overview

Amazon ElastiCache in-transit encryption is a feature that allows you to increase the security of your data at its most vulnerable points—when it is in transit from one location to another. Because there is some processing needed to encrypt and decrypt the data at the endpoints, enabling in-transit encryption can have some performance impact. You should benchmark your data with and without in-transit encryption to determine the performance impact for your use cases.

ElastiCache in-transit encryption implements the following features:

- **Encrypted client connections**—client connections to cache nodes are TLS encrypted.
- **Encrypted server connections—**data moving between nodes in a cluster is encrypted.
- **Server authentication**—clients can authenticate that they are connecting to the right server.
- **Client authentication**—using the Valkey and Redis OSS AUTH feature, the server can authenticate the clients.

> **Note**
>
> ElastiCache does not support mTLS (mutual TLS).

> **Certificate Transparency logging**
>
> ElastiCache provisions a TLS certificate that includes the cluster name for the endpoint of each cluster that has in-transit encryption enabled. AWS Certificate Manager records publicly trusted TLS certificates in public, append-only Certificate Transparency logs. As a result, the name of a cluster with in-transit encryption enabled appears in public Certificate Transparency logs. Don't include confidential or sensitive information in cluster names. For more information about Certificate Transparency logging, see Certificate Transparency logging in the *AWS Certificate Manager User Guide*.

### In-transit encryption conditions (Valkey and Redis OSS)

The following constraints on Amazon ElastiCache in-transit encryption should be kept in mind when you plan your node-based cluster implementation:

- In-transit encryption is supported on replication groups running Valkey and Redis OSS.
- Modifying the in-transit encryption setting, for an existing cluster, is supported on replication groups running Valkey 7.2 and later, and Redis OSS version 7 and later.

  If your cluster runs an engine version that doesn't support this modification, the in-transit encryption option appears disabled (greyed out) in the AWS Management Console when you attempt to modify the cluster. To enable in-transit encryption in this case, you must upgrade your engine version to Valkey 7.2 or later, or Redis OSS version 7 or later.
- In-transit encryption is supported only for replication groups running in an Amazon VPC.
- In-transit encryption is enabled by explicitly setting the parameter `TransitEncryptionEnabled` to `true`.
- Ensure that your caching client supports TLS connectivity and that you have enabled it in client configuration.
- Starting April 28, 2026, AWS will update the minimum supported TLS version to 1.2 on ElastiCache for Valkey version 7.2 and above, and ElastiCache for Redis OSS version 6 and above. Customers must update their client software before that date. This update helps you meet security, compliance, and regulatory needs.

> **Note**
>
> Caches with a public endpoint require TLS 1.3. TLS 1.2 and earlier are not supported for public endpoint connections.

### In-transit encryption conditions (Memcached)

The following constraints on Amazon ElastiCache in-transit encryption should be kept in mind when you plan your node-based cluster implementation:

- In-transit encryption is supported on clusters running Memcached versions 1.6.12 and later.
- In-transit encryption supports Transport Layer Security (TLS) versions 1.2 and 1.3.
- In-transit encryption is supported only for clusters running in an Amazon VPC.

- In-transit encryption is enabled by explicitly setting the parameter `TransitEncryptionEnabled` to `true`.
- You can enable in-transit encryption on a cluster only when creating the cluster. You cannot toggle in-transit encryption on and off by modifying a cluster.
- Ensure that your caching client supports TLS connectivity and that you have enabled it in client configuration.

### In-transit encryption best practices

- Because of the processing required to encrypt and decrypt the data at the endpoints, implementing in-transit encryption can reduce performance. Benchmark in-transit encryption compared to no encryption on your own data to determine its impact on performance for your implementation.
- Because creating new connections can be expensive, you can reduce the performance impact of in-transit encryption by persisting your TLS connections.
- Don't include confidential or sensitive information in cluster names. When you enable in-transit encryption, the cluster's TLS certificate includes the cluster name, and AWS Certificate Manager records that certificate in public Certificate Transparency logs.

### Further Valkey and Redis OSS options

For further information on options available for Valkey and Redis OSS, see the following links.

- At-Rest Encryption in ElastiCache
- Authenticating with the Valkey and Redis OSS AUTH command
- Role-Based Access Control (RBAC)
- Amazon VPCs and ElastiCache security
- Identity and Access Management for Amazon ElastiCache

### Enabling in-transit encryption for Memcached

To enable in-transit encryption when creating a Memcached cluster using the AWS Management Console, make the following selections:

- Choose Memcached as your engine.
- Choose engine version 1.6.12 or later.

- Under **Encryption in transit**, choose **Enable**.

For the step-by-step process, see Creating a cluster for Valkey or Redis OSS.

**Enabling in-transit encryption**

All serverless caches have in-transit encryption enabled. On a node-based cluster, you can enable in-transit encryption using the AWS Management Console, the AWS CLI, or the ElastiCache API.

**Enabling in-transit encryption using the AWS Management Console**

**Enabling in-transit encryption for a new node-based cluster using the AWS Management Console**

When designing your own cluster, 'Dev/Test' and 'Production' configurations with the 'Easy create' method have in-transit encryption enabled. When choosing configuration yourself, make the following selections:

- Click the checkbox next to **Enable** for the **Encryption in transit** option.

For the step-by-step process, see the following:

- Creating a Valkey (cluster mode disabled) cluster (Console)
- Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console)

**Enabling in-transit encryption for an existing node-based cluster using the AWS Management Console**

Enabling encryption in transit, is a two-step process, you must first set the transit encryption mode to `preferred`. This mode allows your Valkey or Redis OSS clients to connect using both encrypted and unencrypted connections. After you migrate all your Valkey or Redis OSS clients to use encrypted connections, you can then modify your cluster configuration to set the transit encryption mode to `required`. Setting the transit encryption mode to `required` will drop all unencrypted connections and will allow encrypted connections only.

**Set your Transit encryption mode to Preferred**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. Choose **Valkey caches** or **Redis OSS caches** from the ElastiCache **Resources** listed on the navigation pane, present on the left hand.
3. Choose the cache you want to update.
4. Choose the **Actions** dropdown, then choose **Modify**.
5. Choose **Enable** under **Encryption in transit** in the **Security** section.
6. Choose **Preferred** as the **Transit encryption mode**.
7. Choose **Preview changes** and save your changes.

After you migrate all your Valkey or Redis OSS clients to use encrypted connections:

**Set your Transit encryption mode to Required**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. Choose **Valkey caches** or **Redis OSS caches** from the ElastiCache **Resources** listed on the navigation pane, present on the left hand.
3. Choose the cache you want to update.
4. Choose the **Actions** dropdown, then choose **Modify**.
5. Choose **Required** as the **Transit encryption mode**, in the **Security** section.
6. Choose **Preview changes** and save your changes.

### Enabling in-transit encryption using the AWS CLI

To enable in-transit encryption when creating a Valkey or Redis OSS replication group using the AWS CLI, use the parameter `transit-encryption-enabled`.

### Enabling in-transit encryption on a new node-based cluster for Valkey or Redis OSS (Cluster Mode Disabled) (CLI)

Use the AWS CLI operation `create-replication-group` and the following parameters to create a Valkey or Redis OSS replication group with replicas that has in-transit encryption enabled:

**Key parameters:**

- **`--engine`**—Must be `valkey` or `redis`.

- **`--transit-encryption-enabled`**—Required. If you enable in-transit encryption, you must also provide a value for the `--cache-subnet-group` parameter.
- **`--num-cache-clusters`**—Must be at least 1. The maximum value for this parameter is six.

For more information, see the following:

- Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch (AWS CLI)
- create-replication-group

**Enabling in-transit encryption on a new node-based cluster for Valkey or Redis OSS (Cluster Mode Enabled) (CLI)**

Use the AWS CLI operation `create-replication-group` and the following parameters to create a Valkey or Redis OSS (cluster mode enabled) replication group that has in-transit encryption enabled:

**Key parameters:**

- **`--engine`**—Must be `valkey` or `redis`.
- **`--transit-encryption-enabled`**—Required. If you enable in-transit encryption you must also provide a value for the `--cache-subnet-group` parameter.
- Use one of the following parameter sets to specify the configuration of the replication group's node groups:
  - **`--num-node-groups`**—Specifies the number of shards (node groups) in this replication group. The maximum value of this parameter is 500.

    **`--replicas-per-node-group`**—Specifies the number of replica nodes in each node group. The value specified here is applied to all shards in this replication group. The maximum value of this parameter is 5.
  - **`--node-group-configuration`**—Specifies the configuration of each shard independently.

For more information, see the following:

- Creating a Valkey or Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)
- create-replication-group

### Enabling in-transit encryption for an existing cluster using the AWS CLI

Enabling encryption in transit, is a two-step process, you must first set the transit encryption mode to `preferred`. This mode allows your Valkey or Redis OSS clients to connect using both encrypted and unencrypted connections. After you migrate all your Valkey or Redis OSS clients to use encrypted connections, you can then modify your cluster configuration to set the transit encryption mode to `required`. Setting the transit encryption mode to `required` will drop all unencrypted connections and will allow encrypted connections only.

Use the AWS CLI operation `modify-replication-group` and the following parameters to update a Valkey or Redis OSS (cluster mode enabled) replication group that has in-transit encryption disabled.

**To enable in-transit encryption**

1. Set transit-encryption-mode to `preferred`, using the following parameters
   - **`--transit-encryption-enabled`**—Required.
   - **`--transit-encryption-mode`**—Must be set to `preferred`.
2. Set transit-encryption-mode to `required`, using the following parameters:
   - **`--transit-encryption-enabled`**—Required.
   - **`--transit-encryption-mode`**—Must be set to `required`.

### Connecting to ElastiCache (Valkey) or Amazon ElastiCache for Redis OSS with in-transit encryption using valkey-cli

To access data from ElastiCache for Redis OSS caches enabled with in-transit encryption, you use clients that work with Secure Socket Layer (SSL). You can also use valkey-cli with TLS on Amazon Linux 2023. If your client does not support TLS, you can use the `stunnel` command on your client host to create an SSL tunnel to the Redis OSS nodes.

**Encrypted connection with Linux**

To use valkey-cli to connect to a Valkey or Redis OSS cluster that has in-transit encryption enabled on Amazon Linux 2023, follow these steps.

1. At the command prompt of your EC2 instance, install the Valkey package. The valkey-cli utility included in this package is built with TLS support.

```
sudo dnf install valkey -y
```

2. Confirm that the utility is installed.

```
valkey-cli --version
```

3. To connect to a cluster with encryption and authentication enabled, enter this command. The `--askpass` option prompts you for the password instead of taking it on the command line, which keeps it out of your shell history and out of the process list.

```
valkey-cli -h Primary or Configuration Endpoint --tls --askpass -p 6379
```

**Note**

If you are connecting to a cluster-mode enabled cache using the Configuration Endpoint, add the `-c` flag to enable cluster mode in the client. This allows the client to follow `MOVED` and `ASK` redirections automatically:

```
valkey-cli -c -h Configuration Endpoint --tls --askpass -p 6379
```

**Note**

If you install the redis6 package on Amazon Linux 2023 instead, use the command `redis6-cli` in place of `valkey-cli`:

```
redis6-cli -h Primary or Configuration Endpoint --tls -p 6379
```

**Note**

If your Linux distribution doesn't provide a Valkey package, you can build the utility from source instead. Install `gcc`, `make`, and the OpenSSL development package, download a current source release as described in Install Valkey on the Valkey website, and then run `make valkey-cli BUILD_TLS=yes`. A client that you build from

source doesn't receive package updates, so you must update it yourself as new releases become available.

### Encrypted connection with stunnel

To use valkey-cli to connect to a Redis OSS cluster enabled with in-transit encryption using stunnel, follow these steps.

1. Use SSH to connect to your client and install `stunnel`.

```
sudo yum install stunnel
```

2. Run the following command to create and edit file `'/etc/stunnel/valkey-cli.conf'` simultaneously to add a ElastiCache for Redis OSS cluster endpoint to one or more connection parameters, using the provided output below as template.

```
vi /etc/stunnel/valkey-cli.conf


fips = no
setuid = root
setgid = root
pid = /var/run/stunnel.pid
debug = 7
delay = yes
options = NO_SSLv2
options = NO_SSLv3
[valkey-cli]
   client = yes
   accept = 127.0.0.1:6379
   connect = primary.ssltest.wif01h.use1.cache.amazonaws.com:6379
[valkey-cli-replica]
   client = yes
   accept = 127.0.0.1:6380
   connect = ssltest-02.ssltest.wif01h.use1.cache.amazonaws.com:6379
```

In this example, the config file has two connections, the `valkey-cli` and the `valkey-cli-replica`. The parameters are set as follows:

- **client** is set to yes to specify this stunnel instance is a client.

- **accept** is set to the client IP. In this example, the primary is set to the Redis OSS default 127.0.0.1 on port 6379. The replica must call a different port and set to 6380. You can use ephemeral ports 1024–65535. For more information, see Ephemeral ports in the *Amazon VPC User Guide.*
- **connect** is set to the Redis OSS server endpoint. For more information, see Finding connection endpoints in ElastiCache.

3. Start `stunnel`.

```
sudo stunnel /etc/stunnel/valkey-cli.conf
```

Use the `netstat` command to confirm that the tunnels started.

```
sudo netstat -tulnp | grep -i stunnel

tcp        0      0 127.0.0.1:6379              0.0.0.0:*                   LISTEN
     3189/stunnel
tcp        0      0 127.0.0.1:6380              0.0.0.0:*                   LISTEN
     3189/stunnel
```

4. Connect to the encrypted Redis OSS node using the local endpoint of the tunnel.

- If no AUTH password was used during ElastiCache for Redis OSS cluster creation, this example uses the valkey-cli to connect to the ElastiCache for Redis OSS server using complete path for valkey-cli, on Amazon Linux:

```
/home/ec2-user/redis-7.2.5/src/valkey-cli -h localhost -p 6379
```

If AUTH password was used during Redis OSS cluster creation, this example uses valkey-cli to connect to the Redis OSS server using complete path for valkey-cli, on Amazon Linux:

```
 /home/ec2-user/redis-7.2.5/src/valkey-cli -h localhost -p 6379 -a my-secret-
password
```

OR

- Change directory to redis-7.2.5 and do the following:

If no AUTH password was used during ElastiCache for Redis OSS cluster creation, this example uses the valkey-cli to connect to the ElastiCache for Redis OSS server using complete path for valkey-cli, on Amazon Linux:

```
src/valkey-cli -h localhost -p 6379
```

If AUTH password was used during Redis OSS cluster creation, this example uses valkey-cli to connect to the Valkey or Redis OSS server using complete path for valkey-cli, on Amazon Linux:

```
src/valkey-cli -h localhost -p 6379 -a my-secret-password
```

This example uses Telnet to connect to the Valkey Redis OSS server.

```
telnet localhost 6379

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
auth MySecretPassword
+OK
get foo
$3
bar
```

5. To stop and close the SSL tunnels, `pkill` the stunnel process.

```
sudo pkill stunnel
```

**Enabling in-transit encryption on a node-based Redis OSS cluster using Python**

The following guide will demonstrate how to enable in-transit encryption on a Redis OSS 7.0 cluster that was originally created with in-transit encryption disabled. TCP and TLS clients will continue communicating with the cluster during this process without downtime.

Boto3 will get the credentials it needs (`aws_access_key_id`, `aws_secret_access_key`, and `aws_session_token`) from the environment variables. Those credentials will be pasted in

advance in the same bash terminal where we will run `python3` to process the Python code shown in this guide. The code in the example below was process from an EC2 instance that was launched in the same VPC that will be used to create the ElastiCache Redis OSS Cluster in it.

**Note**

- The following examples use the boto3 SDK for ElastiCache management operations (cluster or user creation) and redis-py/redis-py-cluster for data handling.
- You must use at least boto3 version (=~) 1.26.39 to use the online TLS migration with the cluster modification API.
- ElastiCache supports online TLS migration only for clusters with Valkey version 7.2 and above or Redis OSS version 7.0 or above. So if you have a cluster running a Redis OSS version earlier than 7.0, you'll need to upgrade the Redis OSS version of your cluster. For more information on version differences, see Major engine version behavior and compatibility differences with Redis OSS.

**Topics**


- Define the string constants that will launch the ElastiCache Valkey or Redis OSS Cluster
- Define the classes for the cluster configuration
- Define a class that will represent the cluster itself
- (Optional) Create a wrapper class to demo client connection to Valkey or Redis OSS cluster
- Create the main function that demos the process of changing in-transit encryption configuration


### Define the string constants that will launch the ElastiCache Valkey or Redis OSS Cluster

First, let's define some simple Python string constants that will hold the names of the AWS entities required to create the ElastiCache cluster such as `security-group`, `Cache Subnet group`, and a `default parameter group`. All of these AWS entities must be created in advance in your AWS account in the Region you are willing to use.

```
#Constants definitions
SECURITY_GROUP = "sg-0492aa0a29c558427"
CLUSTER_DESCRIPTION = "This cluster has been launched as part of the online TLS
 migration user guide"
EC_SUBNET_GROUP = "client-testing"
```

```
DEFAULT_PARAMETER_GROUP_REDIS_7_CLUSTER_MODE_ENABLED = "default.redis7.cluster.on"
```

### Define the classes for the cluster configuration

Now, let's define some simple Python classes that will represent a configuration of a cluster, which will hold metadata about the cluster such as the Valkey or Redis OSS version, the instance type, and whether in-transit encryption (TLS) is enabled or disabled.

```
#Class definitions

class Config:
    def __init__(
        self,
        instance_type: str = "cache.t4g.small",
        version: str = "7.0",
        multi_az: bool = True,
        TLS: bool = True,
        name: str = None,
    ):
        self.instance_type = instance_type
        self.version = version
        self.multi_az = multi_az
        self.TLS = TLS
        self.name = name or f"tls-test"

    def create_base_launch_request(self):
        return {
            "ReplicationGroupId": self.name,
            "TransitEncryptionEnabled": self.TLS,
            "MultiAZEnabled": self.multi_az,
            "CacheNodeType": self.instance_type,
            "Engine": "redis",
            "EngineVersion": self.version,
            "CacheSubnetGroupName": EC_SUBNET_GROUP ,
            "CacheParameterGroupName":
 DEFAULT_PARAMETER_GROUP_REDIS_7_CLUSTER_MODE_ENABLED ,
            "ReplicationGroupDescription": CLUSTER_DESCRIPTION,
            "SecurityGroupIds": [SECURITY_GROUP],
        }

class ConfigCME(Config):
    def __init__(
        self,
```

```
        instance_type: str = "cache.t4g.small",
        version: str = "7.0",
        multi_az: bool = True,
        TLS: bool = True,
        name: str = None,
        num_shards: int = 2,
        num_replicas_per_shard: int = 1,
    ):
        super().__init__(instance_type, version, multi_az, TLS, name)
        self.num_shards = num_shards
        self.num_replicas_per_shard = num_replicas_per_shard

    def create_launch_request(self) -> dict:
        launch_request = self.create_base_launch_request()
        launch_request["NumNodeGroups"] = self.num_shards
        launch_request["ReplicasPerNodeGroup"] = self.num_replicas_per_shard
        return launch_request
```

### Define a class that will represent the cluster itself

Now, let's define some simple Python classes that will represent the ElastiCache Valkey or Redis OSS Cluster itself. This class will have a client field which will hold a boto3 client for ElastiCache management operations such as creating the cluster and querying the ElastiCache API.

```
import botocore.config
import boto3

# Create boto3 client
def init_client(region: str = "us-east-1"):
    config = botocore.config.Config(retries={"max_attempts": 10, "mode": "standard"})
    init_request = dict()
    init_request["config"] = config
    init_request["service_name"] = "elasticache"
    init_request["region_name"] = region
    return boto3.client(**init_request)


class ElastiCacheClusterBase:
    def __init__(self, name: str):
        self.name = name
        self.elasticache_client = init_client()

    def get_first_replication_group(self):
```

```
        return self.elasticache_client.describe_replication_groups(
        ReplicationGroupId=self.name
        )["ReplicationGroups"][0]

    def get_status(self) -> str:
        return self.get_first_replication_group()["Status"]

    def get_transit_encryption_enabled(self) -> bool:
        return self.get_first_replication_group()["TransitEncryptionEnabled"]

    def is_available(self) -> bool:
        return self.get_status() == "available"

    def is_modifying(self) -> bool:
        return self.get_status() == "modifying"

    def wait_for_available(self):
        while True:
            if self.is_available():
                break
            else:
                time.sleep(5)

    def wait_for_modifying(self):
        while True:
            if self.is_modifying():
                break
            else:
                time.sleep(5)

    def delete_cluster(self) -> bool:
        self.elasticache_client.delete_replication_group(
            ReplicationGroupId=self.name, RetainPrimaryCluster=False
        )

    def modify_transit_encryption_mode(self, new_transit_encryption_mode: str):
        # generate api call to migrate the cluster to TLS preffered or to TLS required
            self.elasticache_client.modify_replication_group(
                ReplicationGroupId=self.name,
                TransitEncryptionMode=new_transit_encryption_mode,
                TransitEncryptionEnabled=True,
                ApplyImmediately=True,
            )
        self.wait_for_modifying()
```

```
 class ElastiCacheClusterCME(ElastiCacheClusterBase):
    def __init__(self, name: str):
        super().__init__(name)

    @classmethod
    def launch(cls, config: ConfigCME = None) -> ElastiCacheClusterCME:
        config = config or ConfigCME()
        print(config)
        new_cluster = ElastiCacheClusterCME(config.name)
        launch_request = config.create_launch_request()
        new_cluster.elasticache_client.create_replication_group(**launch_request)
        new_cluster.wait_for_available()
        return new_cluster

    def get_configuration_endpoint(self) -> str:
        return self.get_first_replication_group()["ConfigurationEndpoint"]["Address"]

#Since the code can throw exceptions, we define this class to make the code more
 readable and
#so we won't forget to delete the cluster
class ElastiCacheCMEManager:
    def __init__(self, config: ConfigCME = None):
        self.config = config or ConfigCME()

    def __enter__(self) -> ElastiCacheClusterCME:
        self.cluster = ElastiCacheClusterCME.launch(self.config)
        return self.cluster

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cluster.delete_cluster()
```

### (Optional) Create a wrapper class to demo client connection to Valkey or Redis OSS cluster

Now, let's create a wrapper class for the `redis-py-cluster` client. This wrapper class will support pre-filling the cluster with some keys and then doing random repeated `get` commands.

> **Note**
>
> This is an optional step but it simplifies the code of the main function that comes in a later step.

```
import redis
improt random
from time import perf_counter_ns, time


class DowntimeTestClient:
    def __init__(self, client):
        self.client = client

        # num of keys prefilled
        self.prefilled = 0
        # percent of get above prefilled
        self.percent_get_above_prefilled = 10 # nil result expected when get hit above
 prefilled
        # total downtime in nano seconds
        self.downtime_ns = 0
        # num of success and fail operations
        self.success_ops = 0
        self.fail_ops = 0
        self.connection_errors = 0
        self.timeout_errors = 0


    def replace_client(self, client):
        self.client = client

    def prefill_data(self, timelimit_sec=60):
        end_time = time() + timelimit_sec
        while time() < end_time:
            self.client.set(self.prefilled, self.prefilled)
            self.prefilled += 1

    # unsuccesful operations throw exceptions
    def _exec(self, func):
        try:
            start_ns = perf_counter_ns()
            func()
            self.success_ops += 1
            elapsed_ms = (perf_counter_ns() - start_ns) // 10 ** 6
            # upon succesful execution of func
            # reset random_key to None so that the next command
            # will use a new random key
            self.random_key = None
```

```
        except Exception as e:
            elapsed_ns = perf_counter_ns() - start_ns
            self.downtime_ns += elapsed_ns
            # in case of failure- increment the relevant counters so that we will keep
track
            # of how many connection issues we had while trying to communicate with
            # the cluster.
            self.fail_ops += 1
            if e.__class__ is redis.exceptions.ConnectionError:
                self.connection_errors += 1
            if e.__class__ is redis.exceptions.TimeoutError:
                self.timeout_errors += 1

    def _repeat_exec(self, func, seconds):
        end_time = time() + seconds
        while time() < end_time:
            self._exec(func)

    def _new_random_key_if_needed(self, percent_above_prefilled):
        if self.random_key is None:
            max = int((self.prefilled * (100 + percent_above_prefilled)) / 100)
            return random.randint(0, max)
        return self.random_key

    def _random_get(self):
        key = self._new_random_key_if_needed(self.percent_get_above_prefilled)
        result = self.client.get(key)
        # we know the key was set for sure only in the case key < self.prefilled
        if key < self.prefilled:
            assert result.decode("UTF-8") == str(key)


    def repeat_get(self, seconds=60):
        self._repeat_exec(self._random_get, seconds)

    def get_downtime_ms(self) -> int:
        return self.downtime_ns // 10 ** 6


    def do_get_until(self, cond_check):
        while not cond_check():
            self.repeat_get()
        # do one more get cycle once condition is met
```

```
        self.repeat_get()
```

### Create the main function that demos the process of changing in-transit encryption configuration

Now, let's define the main function, which will do the following:

1. Create the cluster using boto3 ElastiCache client.
2. Initialize the `redis-py-cluster` client that will connect to the cluster with a clear TCP connection without TLS.
3. The `redis-py-cluster` client prefills the cluster with some data.
4. The boto3 client will trigger TLS migration from no-TLS to TLS preferred.
5. While the cluster is being migrated to TLS `Preferred`, the `redis-py-cluster` TCP client will send repeated `get` operations to the cluster until the migration is finished.
6. After the migration to TLS `Preferred` is finished, we will assert that the cluster supports in-transit encryption. Afterwards, we will create a `redis-py-cluster` client that will connect to the cluster with TLS.
7. We will send some `get` commands using the new TLS client and the old TCP client.
8. The boto3 client will trigger TLS migration from TLS `Preferred` to TLS required.
9. While the cluster is being migrated to TLS required, the redis-py-cluster TLS client will send repeated `get` operations to the cluster until the migration is finished.

```
import redis

def init_cluster_client(
    cluster: ElastiCacheClusterCME, prefill_data: bool, TLS: bool = True) ->
 DowntimeTestClient:
    # we must use for the host name the cluster configuration endpoint.
    redis_client = redis.RedisCluster(
        host=cluster.get_configuration_endpoint(), ssl=TLS, socket_timeout=0.25,
 socket_connect_timeout=0.1
    )
    test_client = DowntimeTestClient(redis_client)
    if prefill_data:
        test_client.prefill_data()
    return test_client

if __name__ == '__main__':
```

```
    config = ConfigCME(TLS=False, instance_type="cache.m5.large")

    with ElastiCacheCMEManager(config) as cluster:
        # create a client that will connect to the cluster with clear tcp connection
        test_client_tcp = init_cluster_client(cluster, prefill_data=True, TLS=False)

       # migrate the cluster to TLS Preferred
        cluster.modify_transit_encryption_mode(new_transit_encryption_mode="preferred")

        # do repeated get commands until the cluster finishes the migration to TLS
Preferred
        test_client_tcp.do_get_until(cluster.is_available)

      # verify that in transit encryption is enabled so that clients will be able to
connect to the cluster with TLS
        assert cluster.get_transit_encryption_enabled() == True

      # create a client that will connect to the cluster with TLS connection.
        # we must first make sure that the cluster indeed supports TLS
        test_client_tls = init_cluster_client(cluster, prefill_data=True, TLS=True)

        # by doing get commands with the tcp client for 60 more seconds
      # we can verify that the existing tcp connection to the cluster still works
        test_client_tcp.repeat_get(seconds=60)

        # do get commands with the new TLS client for 60 more seconds
        test_client_tcp.repeat_get(seconds=60)

      # migrate the cluster to TLS required
        cluster.modify_transit_encryption_mode(new_transit_encryption_mode="required")

      # from this point the tcp clients will be disconnected and we must not use them
anymore.
      # do get commands with the TLS client until the cluster finishes migartion to
TLS required mode.
        test_client_tls.do_get_until(cluster.is_available)
```

### Best practices when enabling in-transit encryption

#### Before enabling in-transit encryption: make sure you have proper DNS records handling

> **Note**
>
> We are changing and deleting old endpoints during this process. Incorrect usage of the endpoints can result in the Valkey or Redis OSS client using old and deleted endpoints that will prevent it from connecting to the cluster.

While the cluster is being migrated from no-TLS to TLS-preferred, the old cluster configuration endpoint DNS record is kept and the new cluster configuration endpoint DNS records are being generated in a different format. TLS-enabled clusters use a different format of DNS records than TLS-disabled clusters. ElastiCache will keep both DNS records when a cluster is configured in `encryption mode: Preferred` so that Applications and other Valkey or Redis OSS clients can switch between them. The following changes in the DNS records take place during the TLS-migration process:

#### Description of the changes in the DNS records that take place when enabling in-transit encryption

#### For CME clusters

When a cluster is set to 'transit encryption mode: preferred':

- The original cluster configuration endpoint for no-TLS cluster will remain active. There will be no downtime when cluster is re-configured form TLS encryption mode 'none' to 'preferred'.
- New TLS Valkey or Redis OSS endpoints will be generated when cluster is set to TLS-preferred mode. These new endpoints will resolve to the same IPs as the old ones (non-TLS).
- The new TLS Valkey or Redis OSS configuration endpoint will be exposed in the ElastiCache Console and in the response to `describe-replication-group` API.

When a cluster is set to 'transit encryption mode: required':

- Old non-TLS enabled endpoints will be deleted. There will be no downtime of TLS cluster endpoints.
- You can retrieve a new `cluster-configuration-endpoint` from ElastiCache Console or from the `describe-replication-group` API.

**For CMD clusters with Automatic Failover enabled or Automatic Failover disabled**

When replication group is set to 'transit encryption mode: preferred':

- The original primary endpoint and reader endpoint for non-TLS enabled cluster will remain active.
- New TLS primary and reader endpoints will be generated when cluster is set to TLS `Preferred` mode. This new endpoints will resolve to the same IP(s) as the old ones (non-TLS).
- The new primary endpoint and reader endpoint will be exposed in the ElastiCache Console and in the response to the `describe-replication-group` API.

When replication group is set to 'transit encryption mode: required':

- Old non-TLS primary and reader endpoints will be deleted. There will be no downtime of TLS cluster endpoints.
- You can retrieve new primary and reader endpoints from ElastiCache Console or from the `describe-replication-group` API.

**The suggested usage of the DNS records**

**For CME clusters**

- Use the cluster configuration endpoint instead of per-node DNS records in your application's code. Using per-node DNS names directly is not recommended because during migration they will change and the application code will break connection to the cluster.
- Don't hardcode a cluster configuration endpoint in your application, as it will change during this process.
- Having the cluster configuration endpoint hardcoded in your application is a bad practice, since it can be changed during this process. After the in-transit encryption is completed, query the cluster configuration endpoint with the `describe-replication-group` API (as demonstrated above (in bold)) and use the DNS you get in response from that point on.

**For CMD clusters with Automatic Failover enabled**

- Use the primary endpoint and reader endpoint instead of the per-node DNS names in your application's code since the old per-node DNS names are deleted and new ones are generated when migrating the cluster from no-TLS to TLS-preferred. Using per-node DNS names directly

is not recommended because you might add replicas to your cluster in the future. Also, when Automatic Failover is enabled, the roles of the primary cluster and replicas are changed automatically by the ElastiCache service, using the primary endpoint and reader endpoint is suggested to help you keep track of those changes. Lastly, using the reader endpoint will help you distribute your reads from the replicas equally between the replicas in the cluster.

- Having the primary endpoint and reader endpoint hardcoded in your application is a bad practice since it can be changed during the TLS migration process. After the migration change to TLS-preferred is completed, query the primary endpoint and reader endpoint endpoint with the describe-replication-group API and use the DNS you get in response from this point on. This way you will be able to keep track of changes in endpoints in a dynamic way.

**For CMD cluster with Automatic Failover disabled**

- Use the primary endpoint and reader endpoint instead of the per-node DNS names in your application's code. When Automatic Failover is disabled, scaling, patching, failover, and other procedures that are managed automatically by the ElastiCache service when Automatic Failover is enabled are done by you instead. This makes it easier for you to manually keep track of the different endpoints. Since the old per-node DNS names are deleted and new ones are generated when migrating the cluster from no-TLS to TLS-preferred, do not use the per-node DNS names directly. This is mandatory so that clients will be able to connect to the cluster during the TLS-migration. Also, you'll benefit from evenly spreading the reads between the replicas when using the reader endpoint and keep track of the DNS-records when adding or deleting replicas form the cluster.
- Having the cluster configuration endpoint hardcoded in your application is a bad practice since it can be changed during the TLS migration process.

**During the in-transit encryption: pay attention to when the migration process finishes**

Change of transit encryption mode is not immediate and can take some time. This is especially true for large clusters. Only when the cluster finishes the migration to TLS-preferred is it able to accept and serve both TCP and TLS connections. Therefore, you should not create clients that will try to establish TLS connections to the cluster until the in-transit encryption is completed.

There are several ways to get notified when the in-transit encryption is completed successfully or failed: (Not shown in the code example above):

- Using the SNS service to get a notification when the encryption is completed

- Using the `describe-events` API that will emit an event when the encryption is completed
- Seeing a message in the ElastiCache Console that the encryption is completed

You can also implement logic in your application to know if the encryption is completed. In the example above, we saw several ways to ensure the cluster finishes the migration:

- Waiting until the migration process starts (the cluster status changes to “modifying“), and waiting until the modification is finished (the cluster status changes back to “available“)
- Asserting that the cluster has `transit_encryption_enabled` set to True by querying the `describe-replication-group` API.

**After enabling in-transit encryption: make sure the clients you use are configured properly**

While the cluster is in TLS-preferred mode, your application should open TLS connections to the cluster and only use those connections. This way your application will not experience downtime when enabling in-transit encryption. You can make sure that there are no clearer TCP connections to the Valkey or Redis OSS engine using the info command under the SSL section.

```
# SSL
ssl_enabled:yes
ssl_current_certificate_not_before_date:Mar 20 23:27:07 2017 GMT
ssl_current_certificate_not_after_date:Feb 24 23:27:07 2117 GMT
ssl_current_certificate_serial:D8C7DEA91E684163
tls_mode_connected_tcp_clients:0   (should be zero)
tls_mode_connected_tls_clients:100
```

**Connecting to nodes enabled with in-transit encryption using Openssl (Memcached)**

To access data from ElastiCache for Memcached nodes enabled with in-transit encryption, you need to use clients that work with Secure Socket Layer (SSL). You can also use Openssl s_client on Amazon Linux and Amazon Linux 2.

To use Openssl s_client to connect to a Memcached cluster enabled with in-transit encryption on Amazon Linux 2 or Amazon Linux:

```
/usr/bin/openssl s_client -connect memcached-node-endpoint:memcached-port
```

### Creating a TLS Memcached client using Java

To create a client in TLS mode, do the following to initialize the client with the appropriate SSLContext:

```
import java.security.KeyStore;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;
import net.spy.memcached.AddrUtil;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.MemcachedClient;
public class TLSDemo {
    public static void main(String[] args) throws Exception {
        ConnectionFactoryBuilder connectionFactoryBuilder = new
 ConnectionFactoryBuilder();
        // Build SSLContext
        TrustManagerFactory tmf =
 TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init((KeyStore) null);
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, tmf.getTrustManagers(), null);
        // Create the client in TLS mode
        connectionFactoryBuilder.setSSLContext(sslContext);
        MemcachedClient client = new MemcachedClient(connectionFactoryBuilder.build(),
 AddrUtil.getAddresses("mycluster.fnjyzo.cfg.use1.cache.amazonaws.com:11211"));

        // Store a data item for an hour.
        client.set("theKey", 3600, "This is the data value");
    }
}
```

### Creating a TLS Memcached client using PHP

To create a client in TLS mode, do the following to initialize the client with the appropriate SSLContext:

```
<?php

/**
 * Sample PHP code to show how to create a TLS Memcached client. In this example we
 * will use the Amazon ElastiCache Auto Descovery feature, but TLS can also be
 * used with a Static mode client.
```

```
 * See Using the ElastiCache Cluster Client for PHP (https://docs.aws.amazon.com/
AmazonElastiCache/latest/dg/AutoDiscovery.Using.ModifyApp.PHP.html) for more
 information
 * about Auto Discovery and persistent-id.
 */

/* Configuration endpoint to use to initialize memcached client.
 * this is only an example */
$server_endpoint = "mycluster.fnjyzo.cfg.use1.cache.amazonaws.com";

/* Port for connecting to the cluster.
 * This is only an example     */
$server_port = 11211;

/* Initialize a persistent Memcached client and configure it with the Dynamic client
 mode  */
$tls_client =  new Memcached('persistent-id');
$tls_client->setOption(Memcached::OPT_CLIENT_MODE, Memcached::DYNAMIC_CLIENT_MODE);

/* Add the memcached's cluster server/s */
$tls_client->addServer($server_endpoint, $server_port);

/* Configure the client to use TLS */
if(!$tls_client->setOption(Memcached::OPT_USE_TLS, 1)) {
    echo $tls_client->getLastErrorMessage(), "\n";
    exit(1);
}

/* Set your TLS context configurations values.
 * See MemcachedTLSContextConfig in memcached-api.php for all configurations */
$tls_config = new MemcachedTLSContextConfig();
$tls_config->hostname = '*.mycluster.fnjyzo.use1.cache.amazonaws.com';
$tls_config->skip_cert_verify = false;
$tls_config->skip_hostname_verify = false;

/* Use the created TLS context configuration object to create OpenSSL's SSL_CTX and set
 it to your client.
 * Note:  These TLS context configurations will be applied to all the servers connected
 to this client. */
$tls_client->createAndSetTLSContext((array)$tls_config);

/* test the TLS connection with set-get scenario: */

 /* store the data for 60 seconds in the cluster.
```

```
 * The client will decide which cache host will store this item.
 */
if($tls_client->set('key', 'value', 60)) {
    print "Successfully stored key\n";
} else {
    echo "Failed to set key: ", $tls_client->getLastErrorMessage(), "\n";
    exit(1);
}

/* retrieve the key */
if ($tls_client->get('key') === 'value') {
    print "Successfully retrieved key\n";
} else {
    echo "Failed to get key: ", $tls_client->getLastErrorMessage(), "\n";
    exit(1);
}
```

For more information on using the PHP client, see Installing the ElastiCache cluster client for PHP.

## At-Rest Encryption in ElastiCache

To help keep your data secure, Amazon ElastiCache and Amazon S3 provide different ways to restrict access to data in your cache. For more information, see Amazon VPCs and ElastiCache security and Identity and Access Management for Amazon ElastiCache.

ElastiCache at-rest encryption is a feature to increase data security by encrypting on-disk data. It is always enabled on a serverless cache and on clusters with durability enabled. When enabled, it encrypts the following aspects:

- Disk during sync, backup and swap operations
- Backups stored in Amazon S3
- Multi-AZ transactional durability logs (for clusters with durability enabled)

Data stored on SSDs (solid-state drives) in data tiering enabled clusters is always encrypted.

ElastiCache offers default (service managed) encryption at rest, as well as ability to use your own symmetric customer managed AWS KMS keys in AWS Key Management Service (KMS). When the cache is backed up, under encryption options, choose whether to use the default encryption key or a customer-managed key. For more information, see Enabling At-Rest Encryption.

> **Important**
>
> Enabling At-Rest Encryption on an existing node-based Valkey or Redis OSS cluster involves deleting your existing replication group, **after** running backup and restore on the replication group.

At-rest encryption can be enabled on a cache only when it is created. Because there is some processing needed to encrypt and decrypt the data, enabling at-rest encryption can have a performance impact during these operations. You should benchmark your data with and without at-rest encryption to determine the performance impact for your use cases.

### Topics

- At-Rest Encryption Conditions
- Using customer managed keys from AWS KMS
- Enabling At-Rest Encryption

- Viewing at-rest encryption status
- See Also

### At-Rest Encryption Conditions

The following constraints on ElastiCache at-rest encryption should be kept in mind when you plan your implementation of ElastiCache encryption at-rest:

- At-rest encryption is supported on replication groups running Valkey 7.2 and later, and Redis OSS version 4.0.10 or later.
- At-rest encryption is supported only for replication groups running in an Amazon VPC.
- At-rest encryption is only supported for replication groups running the following node types.
  - R8g, R8gd, R7g, R7gd, R6g, R6gd, R5, R4
  - M8g, M7g, M6g, M5, M4
  - T4g, T3
  - C8gn, C7gn

  For more information, see Supported node types
- At-rest encryption is enabled by setting the parameter `AtRestEncryptionEnabled` to `true`. For Valkey, this parameter defaults to `true` if not specified.
- You can enable at-rest encryption on a replication group only when creating the replication group. You cannot toggle at-rest encryption on and off by modifying a replication group. For information on implementing at-rest encryption on an existing replication group, see Enabling At-Rest Encryption.
- If a cluster is using a data tiering node type, data stored on a solid state drive (SSD) is encrypted whether at-rest encryption is enabled or not.
- If a cluster is using a data tiering node type, data stored on SSD is encrypted with the chosen customer managed AWS KMS key.

- With Memcached, at-rest encryption is supported only on serverless caches.

Implementing at-rest encryption can reduce performance during backup and node sync operations. Benchmark at-rest encryption compared to no encryption on your own data to determine its impact on performance for your implementation.

> **Note**
>
> Clusters with durability enabled (sync or async) always have at-rest encryption enabled. You cannot enable durability without also enabling at-rest encryption.

### Using customer managed keys from AWS KMS

ElastiCache supports symmetric customer managed AWS KMS keys (KMS key) for encryption at rest. Customer-managed KMS keys are encryption keys that you create, own and manage in your AWS account. For more information, see AWS KMS keys in the *AWS Key Management Service Developer Guide*. The keys must be created in AWS KMS before they can be used with ElastiCache.

To learn how to create AWS KMS root keys, see Creating Keys in the *AWS Key Management Service Developer Guide*.

ElastiCache allows you to integrate with AWS KMS. For more information, see Using Grants in the *AWS Key Management Service Developer Guide*. No customer action is needed to enable Amazon ElastiCache integration with AWS KMS.

The `kms:ViaService` condition key limits use of an AWS KMS key (KMS key) to requests from specified AWS services. To use `kms:ViaService` with ElastiCache, include both ViaService names in the condition key value: `elasticache.AWS_region.amazonaws.com` and `dax.AWS_region.amazonaws.com`. For more information, see kms:ViaService.

You can use AWS CloudTrail to track the requests that Amazon ElastiCache sends to AWS Key Management Service on your behalf. All API calls to AWS Key Management Service related to customer managed keys have corresponding CloudTrail logs. You can also see the grants that ElastiCache creates by calling the ListGrants KMS API call.

Once a replication group is encrypted using customer managed key, all backups for the replication group are encrypted as follows:

- Automatic daily backups are encrypted using the customer managed key associated with the cluster.
- Final backup created when replication group is deleted, is also encrypted using the customer managed key associated with the replication group.
- Manually created backups are encrypted by default to use the KMS key associated with the replication group. You may override this by choosing another customer managed key.

- Copying a backup defaults to using a customer managed key associated with the source backup. You may override this by choosing another customer managed key.

> **Note**
>
> - Customer managed keys cannot be used when exporting backups to your selected Amazon S3 bucket. However, all backups exported to Amazon S3 are encrypted using Server side encryption. You may choose to copy the backup file to a new S3 object and encrypt using a customer managed KMS key, copy the file to another S3 bucket that is set up with default encryption using a KMS key or change an encryption option in the file itself.
> - You can also use customer managed keys to encrypt manually-created backups for replication groups that do not use customer managed keys for encryption. With this option, the backup file stored in Amazon S3 is encrypted using a KMS key, even though the data is not encrypted on the original replication group.
>
> Restoring from a backup allows you to choose from available encryption options, similar to encryption choices available when creating a new replication group.

- If you delete the key or disable the key and revoke grants for the key that you used to encrypt a cache, the cache becomes irrecoverable. In other words, it cannot be modified or recovered after a hardware failure. AWS KMS deletes root keys only after a waiting period of at least seven days. After the key is deleted, you can use a different customer managed key to create a backup for archival purposes.
- If you delete the key, disable the key, or revoke grants for the key that you used to encrypt a durability-enabled cluster, the cluster is set to a failed state within 12 hours.
- Automatic key rotation preserves the properties of your AWS KMS root keys, so the rotation has no effect on your ability to access your ElastiCache data. Encrypted Amazon ElastiCache caches don't support manual key rotation, which involves creating a new root key and updating any references to the old key. To learn more, see Rotating AWS KMS keys in the *AWS Key Management Service Developer Guide*.
- Encrypting an ElastiCache cache using KMS key requires one grant per cache. This grant is used throughout the lifespan of the cache. Additionally, one grant per backup is used during backup creation. This grant is retired once the backup is created.

- For more information on AWS KMS grants and limits, see Limits in the *AWS Key Management Service Developer Guide*.

### Enabling At-Rest Encryption

All serverless caches have at-rest encryption enabled.

When creating a node-based cluster, you can enable at-rest encryption by setting the parameter `AtRestEncryptionEnabled` to `true`. You can't enable at-rest encryption on existing replication groups.

You can enable at-rest encryption when you create an ElastiCache cache. You can do so using the AWS Management Console, the AWS CLI, or the ElastiCache API.

When creating a cache, you can pick one of the following options:

- **Default** – This option uses service managed encryption at rest.
- **Customer managed key** – This option allows you to provide the Key ID/ARN from AWS KMS for encryption at rest.

To learn how to create AWS KMS root keys, see Create Keys in the *AWS Key Management Service Developer Guide*

**Contents**


- Enabling At-Rest Encryption Using the AWS Management Console
- Enabling At-Rest Encryption Using the AWS CLI


### Enabling At-Rest Encryption on an existing node-based Valkey or Redis OSS Cluster

You can only enable at-rest encryption when you create a Valkey or Redis OSS replication group. If you have an existing replication group on which you want to enable at-rest encryption, do the following.

**To enable at-rest encryption on an existing replication group**

1. Create a manual backup of your existing replication group. For more information, see Taking manual backups.

2. Create a new replication group by restoring from the backup. On the new replication group, enable at-rest encryption. For more information, see Restoring from a backup into a new cache.
3. Update the endpoints in your application to point to the new replication group.
4. Delete the old replication group. For more information, see Deleting a cluster in ElastiCache or Deleting a replication group.

### Enabling At-Rest Encryption Using the AWS Management Console

#### Enabling At-Rest Encryption on a Serverless Cache (Console)

All serverless caches have at-rest encryption enabled. By default, an AWS-owned KMS key is used to encrypt data. To choose your own AWS KMS key, make the following selections:

- Expand the **Default settings** section.
- Choose **Customize default settings** under **Default settings** section.
- Choose **Customize your security settings** under **Security** section.
- Choose **Customer managed CMK** under **Encryption key** setting.
- Select a key under **AWS KMS key** setting.

#### Enabling At-Rest Encryption on a Node-Based Cluster (Console)

When designing your own cache, 'Dev/Test' and 'Production' configurations with the 'Easy create' method have at-rest encryption enabled using the **Default** key. When choosing configuration yourself, make the following selections:

- Choose version 4.0.10 or later as your engine version.
- Click the checkbox next to **Enable** for the **Encryption at rest** option.
- Choose either a **Default key** or **Customer managed CMK**.

For the step-by-step procedure, see the following:

- Creating a Valkey (cluster mode disabled) cluster (Console)
- Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console)

### Enabling At-Rest Encryption Using the AWS CLI

To enable at-rest encryption when creating a Valkey or Redis OSS cluster using the AWS CLI, use the *--at-rest-encryption-enabled* parameter when creating a replication group.

### Enabling At-Rest Encryption on a Valkey or Redis OSS (Cluster Mode Disabled) Cluster (CLI)

The following operation creates the Valkey or Redis OSS (cluster mode disabled) replication group `my-classic-rg` with three nodes (*--num-cache-clusters*), a primary and two read replicas. At-rest encryption is enabled for this replication group (*--at-rest-encryption-enabled*).

The following parameters and their values are necessary to enable encryption on this replication group:

#### Key Parameters

- **`--engine`**—Must be `valkey` or `redis`.
- **`--engine-version`**—If the engine is Redis OSS, this must be 4.0.10 or later.
- **`--at-rest-encryption-enabled`**—Required to enable at-rest encryption.

#### Example 1: Valkey or Redis OSS (Cluster Mode Disabled) Cluster with Replicas

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id my-classic-rg \
    --replication-group-description "3 node replication group" \
    --cache-node-type cache.m4.large \
    --engine redis \
    --at-rest-encryption-enabled \
    --num-cache-clusters 3
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id my-classic-rg ^
    --replication-group-description "3 node replication group" ^
    --cache-node-type cache.m4.large ^
    --engine redis ^
    --at-rest-encryption-enabled ^
```

```
    --num-cache-clusters 3 ^
```

For additional information, see the following:

- Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch (AWS CLI)
- create-replication-group

### Enabling At-Rest Encryption on a Cluster for Valkey or Redis OSS (Cluster Mode Enabled) (CLI)

The following operation creates the Valkey or Redis OSS (cluster mode enabled) replication group `my-clustered-rg` with three node groups or shards (*--num-node-groups*). Each has three nodes, a primary and two read replicas (*--replicas-per-node-group*). At-rest encryption is enabled for this replication group (*--at-rest-encryption-enabled*).

The following parameters and their values are necessary to enable encryption on this replication group:

**Key Parameters**

- **`--engine`**—Must be `valkey` or `redis`.
- **`--engine-version`**—If the engine is Redis OSS, this must be 4.0.10 or later.
- **`--at-rest-encryption-enabled`**—Required to enable at-rest encryption.
- **`--cache-parameter-group`**—Must be `default-redis4.0.cluster.on` or one derived from it to make this a cluster mode enabled replication group.

### Example 2: A Valkey or Redis OSS (Cluster Mode Enabled) Cluster

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
   --replication-group-id my-clustered-rg \
   --replication-group-description "redis clustered cluster" \
   --cache-node-type cache.m5.large \
   --num-node-groups 3 \
   --replicas-per-node-group 2 \
   --engine redis \
   --engine-version 6.2 \
   --at-rest-encryption-enabled \
```

```
   --cache-parameter-group default.redis6.x.cluster.on
```

For Windows:

```
aws elasticache create-replication-group ^
   --replication-group-id my-clustered-rg ^
   --replication-group-description "redis clustered cluster" ^
   --cache-node-type cache.m5.large ^
   --num-node-groups 3 ^
   --replicas-per-node-group 2 ^
   --engine redis ^
   --engine-version 6.2 ^
   --at-rest-encryption-enabled ^
   --cache-parameter-group default.redis6.x.cluster.on
```

For additional information, see the following:

- Creating a Valkey or Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)
- create-replication-group

### Viewing at-rest encryption status

You can check the at-rest encryption status of an existing cluster using the `StorageEncryptionType` field returned by the describe API calls. This field is available on both node-based replication groups and serverless caches.

`StorageEncryptionType` values:

| Value | Meaning |
|---|---|
| none | At-rest encryption is not enabled |
| `sse-elasticache` | Encrypted using a service-managed AWS KMS key |
| `sse-kms` | Encrypted using a customer-managed AWS KMS key |

### Checking status for a node-based replication group (AWS CLI)

```
aws elasticache describe-replication-groups \
```

```
    --replication-group-id my-replication-group \
    --query 'ReplicationGroups[0].StorageEncryptionType'
```

Example output:

```
"sse-elasticache"
```

**Checking status for a serverless cache (AWS CLI)**

```
aws elasticache describe-serverless-caches \
    --serverless-cache-name my-serverless-cache \
    --query 'ServerlessCaches[0].StorageEncryptionType'
```

Example output:

```
"sse-elasticache"
```

**See Also**

- Amazon VPCs and ElastiCache security
- Identity and Access Management for Amazon ElastiCache

## Authentication and Authorization

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. ElastiCache supports authenticating users using IAM and the Valkey and Redis OSS AUTH command, and authorizing user operations using Role-Based Access Control (RBAC).

**Topics**

- Role-Based Access Control (RBAC)
- Authenticating with the Valkey and Redis OSS AUTH command
- Migrating from password-based authentication (AUTH) to IAM authentication
- Disabling access control on an ElastiCache Valkey or Redis OSS cache

### Role-Based Access Control (RBAC)

With the Valkey and Redis OSS AUTH command as described in Authenticating with the Valkey and Redis OSS AUTH command you can use Role-Based Access Control (RBAC). RBAC is also the only

way to control access to serverless caches. This is available for Valkey 7.2 and onward, and Redis OSS 6.0 to 7.2.

RBAC enables you to:

- Control cache access through user groups. These user groups are designed as a way to organize access to caches.
- With *authN*, have per user passwords as opposed to per cluster auth tokens.
- With *authZ*, have fine-grained user permissions.
- Base your cluster access on ACLs.

Unlike Valkey and Redis OSS AUTH, where all authenticated clients have full cache access if their token is authenticated, RBAC enables you to assign users to sets depending on the users' desired roles. These sets are designed as a way to organize access to caches.

With RBAC, you create users and assign them specific permissions by using an access string, as described following. You assign the users to sets aligned with a specific role (administrators, human resources) that are then deployed to one or more ElastiCache caches. By doing this, you can establish security boundaries between clients using the same Valkey or Redis OSS cache or caches, and prevent clients from accessing each other's data.

RBAC is designed to support the introduction of ACL in Redis OSS 6. When you use RBAC with your ElastiCache Valkey or Redis OSS cache, there are some limitations:

- A user group configured for the "VALKEY" engine can only contain users who are using an authentication mechanism (either password or IAM). This means all users with the engine "VALKEY", and any other users with the engine "Redis" who have their setup configured to authenticate with password or IAM, can be in this user group.
- When using RBAC with Valkey clusters, both user groups with engine "VALKEY" and with engine "REDIS" can be used.
- When using RBAC with Redis OSS clusters, only user groups with the engine "REDIS" can be used.
- You can't specify passwords in an access string. You set passwords with CreateUser or ModifyUser calls.
- For user rights, you enable or disable users with the on and `off` as a part of the access string. If neither is specified in the access string, the user is assigned `off` and doesn't have access rights to the cache.

- You can't use forbidden and renamed commands as part of the access string. If you specify a forbidden or a renamed command, an exception will be thrown. If you want to use access control lists (ACLs) for a renamed command, specify the original name of the command, in other words the name of the command before it was renamed.
- You can't use the `reset` command as a part of an access string. You specify passwords with API parameters, and ElastiCache for Valkey and Redis OSS manages passwords. Thus, you can't use `reset` because it would remove all passwords for a user.
- Redis OSS 6 introduces the ACL LIST command. This command returns a list of users along with the ACL rules applied to each user. ElastiCache supports the `ACL LIST` command, but does not include support for password hashes as Redis OSS does. With ElastiCache, you can use the DescribeUsers operation to get similar information, including the rules contained within the access string. However, DescribeUsers doesn't retrieve a user password.
- Other read-only commands supported by ElastiCache for Valkey and Redis OSS include ACL WHOAMI, ACL USERS, and ACL CAT. ElastiCache for Valkey and Redis OSS doesn't support any other write-based ACL commands.
- The following limits apply:

| Resource | Maximum allowed |
|---|---|
| Users per user group | 100 |
| Number of users | 1000 |
| Number of user groups | 100 |
| User ID length | 40 characters by default |
| User group ID length | 40 characters by default |

**RBAC with Valkey**

When using Role Based Access Control with Valkey, users and user groups are made with the "VALKEY" engine type. This is recommended, as by default Valkey with RBAC provides increased security compared to Redis OSS. Both provisioned and serverless Valkey clusters support VALKEY user and user group associations.

Key features of Valkey Access Control include:

- Valkey users are restricted to Valkey user group associations only.
- Valkey user groups can contain Valkey users, and Redis OSS users who are either password protected or IAM auth enabled.
- Valkey users must use either password protection or IAM authentication.
- VALKEY user groups can only be associated to VALKEY clusters
- There is no default user requirement. When the Valkey user group is attached to clusters, the default user requirement is automatically disabled. Customers will see that default user is turned off when using the ACL LIST command.

More information on using RBAC with ElastiCache for Valkey and Redis OSS follows.

**Topics**


- Specifying Permissions Using an Access String
- Applying RBAC to a Cache for ElastiCache for Valkey or Redis OSS
- Migrating from AUTH to RBAC
- Migrating from RBAC to AUTH
- Automatically rotating passwords for users
- Authenticating with IAM in ElastiCache


### Specifying Permissions Using an Access String

To specify permissions to an ElastiCache Valkey or Redis OSS cache, you create an access string and assign it to a user through either the AWS CLI or AWS Management Console.

Access strings are defined as a list of space-delimited rules which are applied on the user. They define which commands a user can execute and which keys a user can operate on. In order to execute a command, a user must have access to the command being executed and all keys being accessed by the command. Rules are applied from left to right cumulatively, and a simpler string may be used instead of the one provided if there are redundancies in the string provided.

For information about the syntax of the ACL rules, see ACL.

In the following example, the access string represents an active user with access to all available keys and commands.

```
on ~* +@all
```

The access string syntax is broken down as follows:

- on – The user is an active user.
- ~* – Access is given to all available keys.
- +@all – Access is given to all available commands.

The preceding settings are the least restrictive. You can modify these settings to make them more secure.

In the following example, the access string represents a user with access restricted to read access on keys that start with “app::” keyspace

```
on ~app::* -@all +@read
```

You can refine these permissions further by listing commands the user has access to:

+*command1* – The user's access to commands is limited to *command1*.

+@category – The user's access is limited to a category of commands.

For information on assigning an access string to a user, see Creating Users and User Groups with the Console and CLI.

If you are migrating an existing workload to ElastiCache, you can retrieve the access string by calling `ACL LIST`, excluding the user and any password hashes.

For Redis OSS version 6.2 and above the following access string syntax is also supported:

- &* – Access is given to all available channels.

For Redis OSS version 7.0 and above the following access string syntax is also supported:

- | – Can be used for blocking subcommands (e.g "-config|set").
- %R~<pattern> – Add the specified read key pattern. This behaves similar to the regular key pattern but only grants permission to read from keys that match the given pattern. See key permissions for more information.
- %W~<pattern> – Add the specified write key pattern. This behaves similar to the regular key pattern but only grants permission to write to keys that match the given pattern. See ACL key permissions for more information.

- `%RW~<pattern>` – Alias for `~<pattern>`.
- `(<rule list>)` – Create a new selector to match rules against. Selectors are evaluated after the user permissions, and are evaluated according to the order they are defined. If a command matches either the user permissions or any selector, it is allowed. See ACL selectors more information.
- `clearselectors` – Delete all of the selectors attached to the user.

For Valkey version 9.1 and above the following access string syntax is also supported:

- `db=<id>[,<id>...]` – Restricts the user to specific numbered databases. The user can only run commands against the specified databases. Without this rule, a user has access to all databases.

### Applying RBAC to a Cache for ElastiCache for Valkey or Redis OSS

To use ElastiCache for Valkey or Redis OSS RBAC, you take the following steps:

1. Create one or more users.
2. Create a user group and add users to the group.
3. Assign the user group to a cache that has in-transit encryption enabled.

Caches with a public endpoint require every user in the associated user group to use IAM authentication.

These steps are described in detail as follows.

**Topics**

- Creating Users and User Groups with the Console and CLI
- Managing User Groups with the Console and CLI
- Assigning User Groups to Serverless Caches
- Assigning User Groups to Replication Groups

### Creating Users and User Groups with the Console and CLI

The user information for RBAC users is a user ID, user name, and optionally a password and an access string. The access string provides the permission level on keys and commands. The user ID is unique to the user, and the user name is what is passed to the engine.

A user ID can be up to 40 characters long by default. It must begin with a letter, can contain only ASCII letters, digits, and hyphens, and can't contain two consecutive hyphens. A user group ID follows the same rules.

For an IAM-enabled user, the user name must match the user ID, so the same length limit applies to the user name. For more information, see Authenticating with IAM in ElastiCache.

Make sure that the user permissions you provide make sense with the intended purpose of the user group. For example, if you create a user group called `Administrators`, any user you add to that group should have its access string set to full access to keys and commands. For users in an `e-commerce` user group, you might set their access strings to read-only access.

ElastiCache automatically configures a default user with user ID and user name `"default"` and adds it to all user groups. You can't modify or delete this user. This user is intended for compatibility with the default behavior of previous Redis OSS versions and has an access string that permits it to call all commands and access all keys.

To add proper access control to a cache, replace this default user with a new one that isn't enabled or uses a strong password. To change the default user, create a new user with the user name set to `default`. You can then swap it with the original default user.

The following procedures shows how to swap the original `default` user with another `default` user that has a modified access string.

**To modify the default user on the console**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. Choose **User group management** from the navigation pane.
3. For **User group ID**, choose the ID that you want to modify. Make sure that you choose the link and not the check box.
4. Choose **Modify**.
5. In the **Modify** window, choose **Manage**. For "select the user that you want", select the user with the **User name** as default.
6. Choose **Choose**.
7. Choose **Modify**. When you do this, any existing connections to a cache that the original default user has are terminated.

**To modify the default user with the AWS CLI**

1. Create a new user with the user name `default` by using the following commands.

   For Linux, macOS, or Unix:

```
aws elasticache create-user \
 --user-id "new-default-user" \
 --user-name "default" \
 --engine "VALKEY" \
 --passwords "a-str0ng-pa))word" \
 --access-string "off +get ~keys*"
```

   For Windows:

```
aws elasticache create-user ^
 --user-id "new-default-user" ^
 --user-name "default" ^
 --engine "VALKEY" ^
 --passwords "a-str0ng-pa))word" ^
 --access-string "off +get ~keys*"
```

2. Create a user group and add the user that you created previously.

   For Linux, macOS, or Unix:

```
aws elasticache create-user-group \
  --user-group-id "new-group-2" \
  --engine "VALKEY" \
  --user-ids "new-default-user"
```

   For Windows:

```
aws elasticache create-user-group ^
  --user-group-id "new-group-2" ^
  --engine "VALKEY" ^
  --user-ids "new-default-user"
```

When creating a user, you can set up to two passwords. When you modify a password, any existing connections to caches are maintained.

In particular, be aware of these user password constraints when using RBAC for ElastiCache for Valkey and Redis OSS:

- Passwords must be 16–128 printable characters.
- The following nonalphanumeric characters are not allowed: `, "" / @`.

### Managing Users with the Console and CLI

Use the following procedure to manage users on the console.

**To manage users on the console**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the Amazon ElastiCache dashboard, choose **User management**. The following options are available:

   - **Create user** – When creating a user, you enter a user ID, user name, authentication mode, and access string. The access string sets the permission level for what keys and commands the user is allowed.

     When creating a user, you can set up to two passwords. When you modify a password, any existing connections to caches are maintained.
   - **Modify user** – Enables you to update a user's authentication settings or change its access string.
   - **Delete user** – The account will be removed from any User Groups to which it belongs.

Use the following procedures to manage users with the AWS CLI.

**To modify a user by using the CLI**

- Use the `modify-user` command to update a user's password or passwords or change a user's access permissions.

  When a user is modified, the user groups associated with the user are updated, along with any caches associated with the user group. All existing connections are maintained. The following are examples.

For Linux, macOS, or Unix:

```
aws elasticache modify-user \
  --user-id user-id-1 \
  --access-string "~objects:* ~items:* ~public:*" \
  --authentication-mode Type=iam
```

For Windows:

```
aws elasticache modify-user ^
  --user-id user-id-1 ^
  --access-string "~objects:* ~items:* ~public:*" ^
  --authentication-mode Type=iam
```

> **Note**
>
> We don't recommend using the `nopass` option. If you do, we recommend setting the user's permissions to read-only with access to a limited set of keys.

**To delete a user by using the CLI**

- Use the `delete-user` command to delete a user. The account is deleted and removed from any user groups to which it belongs. The following is an example.

  For Linux, macOS, or Unix:

  ```
  aws elasticache delete-user \
    --user-id user-id-2
  ```

  For Windows:

  ```
  aws elasticache delete-user ^
    --user-id user-id-2
  ```

To see a list of users, call the describe-users operation.

```
aws elasticache describe-users
```

### Managing User Groups with the Console and CLI

You can create user groups to organize and control access of users to one or more caches, as shown following.

Use the following procedure to manage user groups using the console.

**To manage user groups using the console**

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the Amazon ElastiCache dashboard, choose **User group management**.

   The following operations are available to create new user groups:

   - **Create** – When you create a user group, you add users and then assign the user groups to caches. For example, you can create an `Admin` user group for users who have administrative roles on a cache.

     > **Important**
     >
     > If you are not using a Valkey or Redis OSS user group then you must include a default user when creating a user group.

   - **Add Users** – Add users to the user group.
   - **Remove Users** – Remove users from the user group. When users are removed from a user group, any existing connections they have to a cache are terminated.
   - **Delete** – Use this to delete a user group. Note that the user group itself, not the users belonging to the group, will be deleted.

   For existing user groups, you can do the following:

   - **Add Users** – Add existing users to the user group.
   - **Delete Users** – Removes existing users from the user group.

> **Note**
>
> Users are removed from the user group, but not deleted from the system.

Use the following procedures to manage user groups using the CLI.

**To create a new user group and add a user by using the CLI**

- Use the `create-user-group` command as shown following.

  For Linux, macOS, or Unix:

```
aws elasticache create-user-group \
  --user-group-id "new-group-1" \
  --engine "VALKEY" \
  --user-ids user-id-1, user-id-2
```

  For Windows:

```
aws elasticache create-user-group ^
  --user-group-id "new-group-1" ^
  --engine "VALKEY" ^
  --user-ids user-id-1, user-id-2
```

**To modify a user group by adding new users or removing current members by using the CLI**

- Use the `modify-user-group` command as shown following.

  For Linux, macOS, or Unix:

```
aws elasticache modify-user-group --user-group-id new-group-1 \
--user-ids-to-add user-id-3 \
--user-ids-to-remove user-id-2
```

  For Windows:

```
aws elasticache modify-user-group --user-group-id new-group-1 ^
--user-ids-to-add user-id-3 ^
```

```
--user-ids-to-removere user-id-2
```

> **Note**
>
> Any open connections belonging to a user removed from a user group are ended by this command.

**To delete a user group by using the CLI**

- Use the `delete-user-group` command as shown following. The user group itself, not the users belonging to the group, is deleted.

  For Linux, macOS, or Unix:

  ```
  aws elasticache delete-user-group /
     --user-group-id
  ```

  For Windows:

  ```
  aws elasticache delete-user-group ^
     --user-group-id
  ```

To see a list of user groups, you can call the describe-user-groups operation.

```
aws elasticache describe-user-groups \
  --user-group-id test-group
```

### Assigning User Groups to Serverless Caches

After you have created a user group and added users, the final step in implementing RBAC is assigning the user group to a serverless cache.

### Assigning User Groups to Serverless Caches Using the Console

To add a user group to a serverless cache using the AWS Management Console, do the following:

- For cluster mode disabled, see Creating a Valkey (cluster mode disabled) cluster (Console)

- For cluster mode enabled, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console)

### Assigning User Groups to Serverless Caches Using the AWS CLI

The following AWS CLI operation creates a serverless cache using the **user-group-id** parameter with the value *my-user-group-id*. Replace the subnet group `sng-test` with a subnet group that exists.

#### Key Parameters

- **`--engine`** – Must be `VALKEY` or `REDIS`.
- **`--user-group-id`** – This value provides the ID of the user group, comprised of users with specified access permissions for the cache.

For Linux, macOS, or Unix:

```
aws elasticache create-serverless-cache \
    --serverless-cache-name "new-serverless-cache" \
    --description "new-serverless-cache" \
    --engine "VALKEY" \
    --user-group-id "new-group-1"
```

For Windows:

```
aws elasticache create-serverless-cache ^
    --serverless-cache-name "new-serverless-cache" ^
    --description "new-serverless-cache" ^
    --engine "VALKEY" ^
    --user-group-id "new-group-1"
```

The following AWS CLI operation modifies a serverless cache with the **user-group-id** parameter with the value *my-user-group-id*.

For Linux, macOS, or Unix:

```
aws elasticache modify-serverless-cache \
    --serverless-cache-name serverless-cache-1 \
    --user-group-id "new-group-2"
```

For Windows:

```
aws elasticache modify-serverless-cache ^
    --serverless-cache-name serverless-cache-1 ^
    --user-group-id "new-group-2"
```

Note that any modifications made to a cache are updated asynchronously. You can monitor this progress by viewing the events. For more information, see Viewing ElastiCache events.

### Assigning User Groups to Replication Groups

After you have created a user group and added users, the final step in implementing RBAC is assigning the user group to a replication group.

### Assigning User Groups to Replication Groups Using the Console

To add a user group to a replication using the AWS Management Console, do the following:

- For cluster mode disabled, see Creating a Valkey (cluster mode disabled) cluster (Console)
- For cluster mode enabled, see Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console)

### Assigning User Groups to Replication Groups Using the AWS CLI

The following AWS CLI operation creates a replication group that has encryption in transit (TLS) enabled and required, with the **user-group-ids** parameter with the value *new-group-1*. Replace the subnet group `cache-subnet-group` with a subnet group that exists.

#### Key Parameters

- **`--engine`** – Must be `valkey` or `redis`.
- **`--engine-version`** – Must be 6.0 or later.
- **`--transit-encryption-enabled`** – Required for authentication and for associating a user group.
- **`--transit-encryption-mode`** – Must be `required` or omitted for authentication and for associating a user group.
- **`--user-group-ids`** – This value provides the ID of the user group, comprised of users with specified access permissions for the cache.

- **`--cache-subnet-group`** – Required for associating a user group.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id "new-replication-group" \
    --replication-group-description "new-replication-group" \
    --engine "VALKEY" \
    --cache-node-type cache.m5.large \
    --transit-encryption-enabled \
    --user-group-ids "new-group-1" \
    --cache-subnet-group "cache-subnet-group"
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id "new-replication-group" ^
    --replication-group-description "new-replication-group" ^
    --engine "VALKEY" ^
    --cache-node-type cache.m5.large ^
    --transit-encryption-enabled ^
    --user-group-ids "new-group-1" ^
    --cache-subnet-group "cache-subnet-group"
```

The following AWS CLI operation modifies a replication group that has encryption in transit (TLS) enabled and required, with the **user-group-ids** parameter with the value *new-group-2*.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id replication-group-1 \
    --user-group-ids-to-remove "new-group-1" \
    --user-group-ids-to-add "new-group-2"
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id replication-group-1 ^
    --user-group-ids-to-remove "new-group-1" ^
    --user-group-ids-to-add "new-group-2"
```

Note the `PendingChanges` in the response. Any modifications made to a cache are updated asynchronously. You can monitor this progress by viewing the events. For more information, see Viewing ElastiCache events.

**Migrating from AUTH to RBAC**

If you are using AUTH as described in Authenticating with the Valkey and Redis OSS AUTH command and want to migrate to using RBAC, use the following procedures.

Use the following procedure to migrate from AUTH to RBAC using the console.

**To migrate from Valkey or Redis OSS AUTH to RBAC using the console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region where the cache that you want to modify is located.
3. In the navigation pane, choose the engine that runs on the cache that you want to modify.

   A list of the chosen engine's caches appears.
4. In the list of caches, for the cache that you want to modify, choose its name.
5. For **Actions**, choose **Modify**.

   The **Modify** window appears.
6. For **Access control**, choose **User group access control list**.
7. For **User group access control list**, choose a user group.
8. Choose **Preview changes** and then on the next screen, **Modify**.

Use the following procedure to migrate from Valkey or Redis OSS AUTH to RBAC using the CLI.

**To migrate from AUTH to RBAC using the CLI**

- Use the `modify-replication-group` command as shown following.

  For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group --replication-group-id test \
    --auth-token-update-strategy DELETE \
```

```
    --user-group-ids-to-add user-group-1
```

For Windows:

```
aws elasticache modify-replication-group --replication-group-id test ^
    --auth-token-update-strategy DELETE ^
    --user-group-ids-to-add user-group-1
```

### Migrating from RBAC to AUTH

If you are using RBAC and want to migrate to Redis OSS AUTH, see Migrating from RBAC to AUTH.

> **Note**
>
> If you need to disable access control on an ElastiCache cache, you'll need to do it through the AWS CLI. For more information, see the section called "Disabling access control on an ElastiCache Valkey or Redis OSS cache".

### Automatically rotating passwords for users

With AWS Secrets Manager, you can replace hardcoded credentials in your code (including passwords) with an API call to Secrets Manager to retrieve the secret programmatically. This helps ensure that the secret can't be compromised by someone examining your code, because the secret simply isn't there. Also, you can configure Secrets Manager to automatically rotate the secret for you according to a schedule that you specify. This enables you to replace long-term secrets with short-term ones, which helps to significantly reduce the risk of compromise.

Using Secrets Manager, you can automatically rotate your ElastiCache for Redis OSS passwords (that is, secrets) using an AWS Lambda function that Secrets Manager provides.

> **Valkey authentication requirement**
>
> Valkey does not support the `no-password-required` authentication mode. When you use Valkey, create the initial user with a temporary password. Set the access string to `off` (disabled) instead of using the `--no-password` flag shown in Step 1 below.

For more information about AWS Secrets Manager, see What is AWS Secrets Manager?

### How ElastiCache uses secrets

Valkey 7.2 and above have an equivalent feature set as Redis OSS 7.0. In Redis OSS 6, ElastiCache introduced Role-Based Access Control (RBAC) to secure the Valkey or Redis OSS cluster. This feature allows certain connections to be limited in terms of the commands that can be executed and the keys that can be accessed. With RBAC, while the customer creates a user with passwords, the password values need to be manually entered in plaintext and is visible to the operator.

With Secrets Manager, applications fetch the password from Secrets Manager rather than entering them manually and storing them in the application's configuration. For information on how to do this, see How ElastiCache users are associated with the secret.

There is a cost incurred for using secrets. For pricing information, see AWS Secrets Manager Pricing.

### How ElastiCache users are associated with the secret

Secrets Manager will keep a reference for the associated user in the secret's `SecretString` field. There will be no reference to the secret from ElastiCache side.

```
{
    "password": "strongpassword",
    "username": "user1",
    "user_arn": "arn:aws:elasticache:us-east-1:xxxxxxxxxx918:user:user1" //this is the
 bond between the secret and the user
}
```

### Lambda rotation function

To enable Secrets Manager automatic password rotation, you will create a Lambda function that will interact with the modify-user API to update the user's passwords.

For information on how this works, see How rotation works.

> **Note**
>
> For some AWS services, to avoid the confused deputy scenario, AWS recommends that you use both the `aws:SourceArn` and `aws:SourceAccount` global condition keys. However, if you include the `aws:SourceArn` condition in your rotation function policy, the rotation function can only be used to rotate the secret specified by that ARN. We recommend that you include only the context key `aws:SourceAccount` so that you can use the rotation function for multiple secrets.

For any issues you may encounter, see Troubleshoot AWS Secrets Manager rotation.

**How to create an ElastiCache user and associate it with Secrets Manager**

The following steps illustrate how to create a user and associate it with Secrets Manager:

1. **Create an inactive user**

   For Linux, macOS, or Unix:

```
aws elasticache create-user \
 --user-id user1 \
 --user-name user1 \
 --engine "REDIS" \
 --no-password \ // no authentication is required
 --access-string "*off* +get ~keys*" // this disables the user
```

   For Windows:

```
aws elasticache create-user ^
 --user-id user1 ^
 --user-name user1 ^
 --engine "REDIS" ^
 --no-password ^ // no authentication is required
 --access-string "*off* +get ~keys*" // this disables the user
```

   You will see a response similar to the following:

```
{
    "UserId": "user1",
    "UserName": "user1",
    "Status": "active",
    "Engine": "redis",
    "AccessString": "off ~keys* -@all +get",
    "UserGroupIds": [],
    "Authentication": {
        "Type": "no_password"
    },
    "ARN": "arn:aws:elasticache:us-east-1:xxxxxxxxxx918:user:user1"
}
```

2. **Create a Secret**

For Linux, macOS, or Unix:

```
aws secretsmanager create-secret \
--name production/ec/user1 \
--secret-string \
'{
   "user_arn": "arn:aws:elasticache:us-east-1:123456xxxx:user:user1",
    "username":"user1"
 }'
```

For Windows:

```
aws secretsmanager create-secret ^
--name production/ec/user1 ^
--secret-string ^
'{
   "user_arn": "arn:aws:elasticache:us-east-1:123456xxxx:user:user1",
    "username":"user1"
 }'
```

You will see a response similar to the following:

```
{
 "ARN": "arn:aws:secretsmanager:us-east-1:123456xxxx:secret:production/ec/user1-
eaFois",
 "Name": "production/ec/user1",
 "VersionId": "aae5b963-1e6b-4250-91c6-ebd6c47d0d95"
}
```

3. **Configure a Lambda function to rotate your password**

   a. Sign in to the AWS Management Console and open the Lambda console at https://console.aws.amazon.com/lambda/

   b. Choose **Functions** on the navigation pane and then choose the function you created. Choose the function name, not the checkbox to its left.

   c. Choose the **Configuration** tab.

   d. In **General configuration**, choose **Edit** and then set **Timeout** to at least 12 minutes.

   e. Choose **Save**.

   f. Choose **Environment variables** and then set the following:

i. SECRETS_MANAGER_ENDPOINT – https://secretsmanager.**REGION**.amazonaws.com

ii. SECRET_ARN – The Amazon Resource Name (ARN) of the secret you created in Step 2.

iii. USER_NAME – Username of the ElastiCache user,

iv. Choose **Save**.

g. Choose **Permissions**

h. Under **Execution role**, choose the name of the Lambda function role to view on the IAM console.

i. The Lambda function will need the following permission to modify the users and set the password:

ElastiCache

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:DescribeUsers",
                "elasticache:ModifyUser"
            ],
            "Resource": "arn:aws:elasticache:us-
east-1:123456789012:user:user1"
        }
    ]
}
```

Secrets Manager

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:XXXX"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetRandomPassword",
            "Resource": "*"
        }
    ]
}
```

4. Set up Secrets Manager secret rotation

    a. **Using the AWS Management Console, see Set up automatic rotation for AWS Secrets Manager secrets using the console**

    For more information on setting up a rotation schedule, see Schedule expressions in Secrets Manager rotation.

    b. **Using the AWS CLI, see Set up automatic rotation for AWS Secrets Manager using the AWS Command Line Interface**

### Authenticating with IAM in ElastiCache

### Topics

- Overview
- Limitations
- Setup
- Connecting

### Overview

With IAM Authentication you can authenticate a connection to ElastiCache for Valkey or Redis OSS using AWS IAM identities, when your cache is configured to use Valkey or Redis OSS version 7 or above. This allows you to strengthen your security model and simplify many administrative security tasks. You can also use IAM Authentication to configure fine-grained access control for each individual ElastiCache cache and ElastiCache user, following least-privilege permissions principles. IAM Authentication for ElastiCache works by providing a short-lived IAM authentication token instead of a long-lived ElastiCache user password in the Valkey or Redis OSS `AUTH` or `HELLO` command. For more information about the IAM authentication token, refer to the Signature Version 4 signing process in the the AWS General Reference Guide and the code example below.

You can use IAM identities and their associated policies to further restrict Valkey or Redis OSS access. You can also grant access to users from their federated Identity providers directly to Valkey or Redis OSS caches.

To use AWS IAM with ElastiCache, you first need to create an ElastiCache user with authentication mode set to IAM. Then you can create or reuse an IAM identity. The IAM identity needs an associated policy to grant the `elasticache:Connect` action to the ElastiCache cache and ElastiCache user. Once configured, you can create an IAM authentication token using the AWS credentials of the IAM user or role. Finally you need to provide the short-lived IAM authentication token as a password in your Valkey or Redis OSS Client when connecting to your cache. A Valkey or Redis OSS client with support for credentials provider can auto-generate the temporary credentials automatically for each new connection. ElastiCache will perform IAM authentication for connection requests of IAM-enabled ElastiCache users and will validate the connection requests with IAM.

### Limitations

When using IAM authentication, the following limitations apply:

- IAM authentication is available when using ElastiCache for Valkey 7.2 and above or Redis OSS version 7.0 and above.
- IAM authentication requires in-transit encryption (TLS) to be enabled on your cache. For more information, see the section called “In-transit encryption (TLS)”.
- For IAM-enabled ElastiCache users the username and user id properties must be identical.
- The IAM authentication token is valid for 15 minutes. If the connection is re-authenticated with an expired token, the authentication request will be rejected. For long-lived connections, we recommend using a Valkey or Redis OSS client that supports a credentials provider interface to automatically generate fresh tokens before expiry.

- An IAM authenticated connection to ElastiCache for Valkey or Redis OSS will automatically be disconnected after 12 hours. The connection can be prolonged for 12 hours by sending an `AUTH` or `HELLO` command with a new IAM authentication token.
- IAM re-authentication (`AUTH` or `HELLO` commands) is not supported inside `MULTI/EXEC` or Lua script blocks. However, you can run regular data commands inside `MULTI/EXEC` blocks on an IAM-authenticated connection.
- Currently, IAM authentication supports the following global condition context keys:
  - When using IAM authentication with serverless caches with a VPC endpoint, `aws:VpcSourceIp`, `aws:SourceVpc`, `aws:SourceVpce`, `aws:CurrentTime`, `aws:EpochTime`, and `aws:ResourceTag/%s` (from associated serverless caches and users) are supported.
  - When using IAM authentication with replication groups, `aws:SourceIp` and `aws:ResourceTag/%s` (from associated replication groups and users) are supported.

  For more information about global condition context keys, see AWS global condition context keys in the IAM User Guide.

> **Note**
>
> Cache names are converted to lowercase at cache creation time. Ensure authenticating code supplies the cache name in lowercase to avoid authentication errors.

### Setup

To setup IAM authentication:

1. Create a cache

```
aws elasticache create-serverless-cache \
  --serverless-cache-name cache-01  \
  --description "ElastiCache IAM auth application" \
  --engine redis
```

2. Create an IAM trust policy document, as shown below, for your role that allows your account to assume the new role. Save the policy to a file named *trust-policy.json*.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/my-application-role"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

**Note**

Replace *my-application-role* with the IAM role or user that needs to connect to your cache. Use a least-privilege principal rather than the account root to limit which identities can assume this role.

3. Create an IAM policy document, as shown below. Save the policy to a file named *policy.json*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticache:Connect"
      ],
      "Resource" : [
        "arn:aws:elasticache:us-
east-1:123456789012:serverlesscache:cache-01",
        "arn:aws:elasticache:us-east-1:123456789012:user:iam-user-01"
      ]
    }
  ]
```

```
}
```

4. Create an IAM role.

```
aws iam create-role \
--role-name "elasticache-iam-auth-app" \
--assume-role-policy-document file://trust-policy.json
```

5. Create the IAM policy.

```
aws iam create-policy \
  --policy-name "elasticache-allow-all" \
  --policy-document file://policy.json
```

6. Attach the IAM policy to the role.

```
aws iam attach-role-policy \
 --role-name "elasticache-iam-auth-app" \
 --policy-arn "arn:aws:iam::123456789012:policy/elasticache-allow-all"
```

7. Create a new IAM-enabled user.

```
aws elasticache create-user \
  --user-name iam-user-01 \
  --user-id iam-user-01 \
  --authentication-mode Type=iam \
  --engine redis \
  --access-string "on ~* +@all"
```

8. Create a user group and attach the user.

```
aws elasticache create-user-group \
  --user-group-id iam-user-group-01 \
  --engine redis \
  --user-ids default iam-user-01

aws elasticache modify-serverless-cache \
  --serverless-cache-name cache-01  \
  --user-group-id iam-user-group-01
```

> **Note**
>
> Caches with a public endpoint require a user group where all users use IAM authentication. ElastiCache provides a default user (`default.iam-user`) and user group (`default.iam-user-group`) that you can use to get started.

### Connecting

### Connecting with GLIDE

GLIDE 2.2 and later include built-in IAM authentication support. You configure your AWS credentials, and GLIDE handles token generation, caching, and refresh automatically. For installation instructions, see the GLIDE documentation on the GLIDE website.

### Python

```
from glide import (
    GlideClusterClient, GlideClusterClientConfiguration, NodeAddress,
    ServerCredentials, IamAuthConfig, ServiceType,
)

config = GlideClusterClientConfiguration(
    addresses=[NodeAddress("my-cache-x2e9hv.serverless.use1.cache.amazonaws.com",
 6379)],
    credentials=ServerCredentials(
        username="my-iam-user",
        iam_config=IamAuthConfig(
            cluster_name="my-cache",
            service=ServiceType.ELASTICACHE,
            region="us-east-1",
        ),
    ),
    use_tls=True,
)

client = await GlideClusterClient.create(config)
await client.set("key", "value")
result = await client.get("key")
print(result)  # b"value"
```

### Java

```
import glide.api.GlideClusterClient;
import glide.api.models.configuration.IamAuthConfig;
import glide.api.models.configuration.ServiceType;
import glide.api.models.configuration.ServerCredentials;
import glide.api.models.configuration.GlideClusterClientConfiguration;
import glide.api.models.configuration.NodeAddress;

import java.util.List;
import java.util.Collections;

List<NodeAddress> nodes = Collections.singletonList(
    NodeAddress.builder()
        .host("my-cache-x2e9hv.serverless.use1.cache.amazonaws.com")
        .port(6379)
        .build()
);

IamAuthConfig iamConfig = IamAuthConfig.builder()
    .clusterName("my-cache")
    .service(ServiceType.ELASTICACHE)
    .region("us-east-1")
    .build();

GlideClusterClientConfiguration config = GlideClusterClientConfiguration.builder()
    .addresses(nodes)
    .credentials(ServerCredentials.builder()
        .username("my-iam-user")
        .iamConfig(iamConfig)
        .build())
    .useTLS(true)
    .build();

GlideClusterClient client = GlideClusterClient.createClient(config).get();
client.set("key", "value").get();
String result = client.get("key").get();
System.out.println(result);  // "value"
```

### Connecting with the Developer Toolkit for ElastiCache

The Developer Toolkit for ElastiCache (`developer-toolkit-elasticache`) is a standalone Python library and CLI that generates IAM authentication tokens. Use it with a Python Valkey or Redis OSS client library, such as valkey-py.

```
pip install developer-toolkit-elasticache
```

### Python with valkey-py

```
import valkey
from valkey.credentials import CredentialProvider
from developer_toolkit_elasticache import ElastiCacheIAMAuthTokenProvider

class ElastiCacheCredentialProvider(CredentialProvider):
    def __init__(self, auth):
        self._auth = auth
    def get_credentials(self):
        return self._auth.user_id, self._auth.get_token()

auth = ElastiCacheIAMAuthTokenProvider(
    serverless_cache_name="my-cache",
    user_id="my-iam-user",
    region="us-east-1",
)

# For a node-based cluster, use replication_group_id instead:
# auth = ElastiCacheIAMAuthTokenProvider(
#     replication_group_id="my-cluster",
#     user_id="my-iam-user",
#     region="us-east-1",
# )

client = valkey.ValkeyCluster(
    host="my-cache-x2e9hv.serverless.use1.cache.amazonaws.com",
    port=6379,
    ssl=True,
    credential_provider=ElastiCacheCredentialProvider(auth),
)

client.set("key", "value")
result = client.get("key")
print(result)  # b"value"
```

The Developer Toolkit does not automatically refresh tokens. IAM authentication tokens are valid for 15 minutes, so your application must generate a new token before the current token expires.

### CLI with the Developer Toolkit

To generate a token from the command line:

```
developer-toolkit-elasticache generate_iam_auth_token \
    --serverless-cache-name my-cache \
    --user-id my-iam-user \
    --region us-east-1
```

To connect with valkey-cli or redis-cli, set the authentication environment variable and connect:

```
export VALKEYCLI_AUTH=$(developer-toolkit-elasticache generate_iam_auth_token \
    --serverless-cache-name my-cache \
    --user-id my-iam-user \
    --region us-east-1)

valkey-cli -c -h my-cache-x2e9hv.serverless.use1.cache.amazonaws.com \
    -p 6379 \
    --tls \
    --user my-iam-user
```

`VALKEYCLI_AUTH` requires valkey-cli 9.0 or later. For redis-cli, use `REDISCLI_AUTH` instead of `VALKEYCLI_AUTH`.

For source code and additional examples, see the Developer Toolkit for ElastiCache repository on GitHub.

**Connect with token as password**

You first need to generate the short-lived IAM authentication token using an AWS SigV4 pre-signed request. After that you provide the IAM authentication token as a password when connecting to a Valkey or Redis OSS cache, as shown in the example below.

```
String userId = "insert user id";
String cacheName = "insert cache name";
boolean isServerless = true;
String region = "insert region";

// Create a default AWS Credentials provider.
// This will look for AWS credentials defined in environment variables or system
 properties.
AwsCredentialsProvider awsCredentialsProvider = DefaultCredentialsProvider.create();

// Create an IAM authentication token request and signed it using the AWS credentials.
```

```
// The pre-signed request URL is used as an IAM authentication token for ElastiCache
 with Redis OSS.
IAMAuthTokenRequest iamAuthTokenRequest = new IAMAuthTokenRequest(userId, cacheName,
 region, isServerless);
String iamAuthToken =
 iamAuthTokenRequest.toSignedRequestUri(awsCredentialsProvider.resolveCredentials());

// Construct Redis OSS URL with IAM Auth credentials provider
RedisURI redisURI = RedisURI.builder()
    .withHost(host)
    .withPort(port)
    .withSsl(ssl)
    .withAuthentication(userId, iamAuthToken)
    .build();

// Create a new Lettuce Redis OSS client
RedisClient client = RedisClient.create(redisURI);
client.connect();
```

Below is the definition for `IAMAuthTokenRequest`.

```
public class IAMAuthTokenRequest {
    private static final SdkHttpMethod REQUEST_METHOD = SdkHttpMethod.GET;
    private static final String REQUEST_PROTOCOL = "http://";
    private static final String PARAM_ACTION = "Action";
    private static final String PARAM_USER = "User";
    private static final String PARAM_RESOURCE_TYPE = "ResourceType";
    private static final String RESOURCE_TYPE_SERVERLESS_CACHE = "ServerlessCache";
    private static final String ACTION_NAME = "connect";
    private static final String SERVICE_NAME = "elasticache";
    private static final Duration TOKEN_EXPIRY_DURATION = Duration.ofSeconds(900);

    private final String userId;
    private final String cacheName;
    private final String region;
    private final boolean isServerless;

    public IAMAuthTokenRequest(String userId, String cacheName, String region, boolean
 isServerless) {
        this.userId = userId;
        this.cacheName = cacheName;
        this.region = region;
        this.isServerless = isServerless;
```

```
    }

    public String toSignedRequestUri(AwsCredentials credentials) {
        SdkHttpFullRequest request = getSignableRequest();
        SdkHttpFullRequest signedRequest = sign(request, credentials);
        return signedRequest.getUri().toString().replace(REQUEST_PROTOCOL, "");
    }

    private SdkHttpFullRequest getSignableRequest() {
        SdkHttpFullRequest.Builder builder = SdkHttpFullRequest.builder()
            .method(REQUEST_METHOD)
            .uri(getRequestUri())
            .appendRawQueryParameter(PARAM_ACTION, ACTION_NAME)
            .appendRawQueryParameter(PARAM_USER, userId);
        if (isServerless) {
            builder.appendRawQueryParameter(PARAM_RESOURCE_TYPE,
 RESOURCE_TYPE_SERVERLESS_CACHE);
        }
        return builder.build();
    }

    private URI getRequestUri() {
        return URI.create(String.format("%s%s/", REQUEST_PROTOCOL, cacheName));
    }

    private SdkHttpFullRequest sign(SdkHttpFullRequest request, AwsCredentials
 credentials) {
        AwsV4HttpSigner signer = AwsV4HttpSigner.create();
        SignedRequest signedRequest = signer.sign(r -> r.identity(credentials)
            .request(request)
            .putProperty(AwsV4HttpSigner.SERVICE_SIGNING_NAME, SERVICE_NAME)
            .putProperty(AwsV4HttpSigner.REGION_NAME, region)
            .putProperty(AwsV4HttpSigner.AUTH_LOCATION,
 AwsV4HttpSigner.AuthLocation.QUERY_STRING)
            .putProperty(AwsV4HttpSigner.EXPIRATION_DURATION, TOKEN_EXPIRY_DURATION)
            .build()
        );
        return (SdkHttpFullRequest) signedRequest.request();
    }
}
```

### Connect with credentials provider

The code below shows how to authenticate with ElastiCache using the IAM authentication credentials provider.

```
String userId = "insert user id";
String cacheName = "insert cache name";
boolean isServerless = true;
String region = "insert region";

// Create a default AWS Credentials provider.
// This will look for AWS credentials defined in environment variables or system
 properties.
AwsCredentialsProvider awsCredentialsProvider = DefaultCredentialsProvider.create();

// Create an IAM authentication token request. Once this request is signed it can be
 used as an
// IAM authentication token for ElastiCache with Redis OSS.
IAMAuthTokenRequest iamAuthTokenRequest = new IAMAuthTokenRequest(userId, cacheName,
 region, isServerless);

// Create a Redis OSS credentials provider using IAM credentials.
RedisCredentialsProvider redisCredentialsProvider = new
 RedisIAMAuthCredentialsProvider(
    userId, iamAuthTokenRequest, awsCredentialsProvider);

// Construct Redis OSS URL with IAM Auth credentials provider
RedisURI redisURI = RedisURI.builder()
    .withHost(host)
    .withPort(port)
    .withSsl(ssl)
    .withAuthentication(redisCredentialsProvider)
    .build();

// Create a new Lettuce Redis OSS client
RedisClient client = RedisClient.create(redisURI);
client.connect();
```

Below is an example of a Lettuce Redis OSS client that wraps the IAMAuthTokenRequest in a credentials provider to auto-generate temporary credentials when needed.

```
public class RedisIAMAuthCredentialsProvider implements RedisCredentialsProvider {
    private static final long TOKEN_EXPIRY_SECONDS = 900;
```

```
    private final AwsCredentialsProvider awsCredentialsProvider;
    private final String userId;
    private final IAMAuthTokenRequest iamAuthTokenRequest;
    private final Supplier<String> iamAuthTokenSupplier;

    public RedisIAMAuthCredentialsProvider(String userId,
        IAMAuthTokenRequest iamAuthTokenRequest,
        AwsCredentialsProvider awsCredentialsProvider) {
        this.userId = userId;
        this.awsCredentialsProvider = awsCredentialsProvider;
        this.iamAuthTokenRequest = iamAuthTokenRequest;
        this.iamAuthTokenSupplier =
 Suppliers.memoizeWithExpiration(this::getIamAuthToken, TOKEN_EXPIRY_SECONDS,
 TimeUnit.SECONDS);
    }

    @Override
    public Mono<RedisCredentials> resolveCredentials() {
        return Mono.just(RedisCredentials.just(userId, iamAuthTokenSupplier.get()));
    }

    private String getIamAuthToken() {
        return
 iamAuthTokenRequest.toSignedRequestUri(awsCredentialsProvider.resolveCredentials());
    }
}
```

### Authenticating with the Valkey and Redis OSS AUTH command

> **Note**
>
> The **AUTH** has been superseded by the section called "Role-Based Access Control (RBAC)". All serverless caches must use RBAC for authentication. With RBAC, you can also use Secrets Manager for automatic password rotation. For more information, see Automatically rotating passwords for users.

Valkey and Redis OSS authentication tokens or passwords enable Valkey and Redis OSS to require a password before allowing clients to run commands, thereby improving data security. The **AUTH** is available for node-based clusters only.

**Topics**


- Overview of AUTH in ElastiCache for Valkey and Redis OSS
- Applying authentication to an ElastiCache for Valkey and Redis OSS cluster
- Modifying the AUTH token on an existing cluster
- Migrating from RBAC to AUTH


### Overview of AUTH in ElastiCache for Valkey and Redis OSS

When you use the **AUTH** with your ElastiCache for Valkey and Redis OSS cluster, there are some refinements.

In particular, be aware of these AUTH token or password constraints when using AUTH:

- Tokens, or passwords, must be 16–128 printable characters.
- Nonalphanumeric characters are restricted to (!, &, #, $, ^, <, >, -).
- AUTH can only be enabled for encryption in-transit enabled Valkey or Redis OSS clusters.

To set up a strong token, we recommend that you follow a strict password policy, such as requiring the following:

- Tokens or passwords must include at least three of the following character types:
  - Uppercase characters
  - Lowercase characters
  - Digits
  - Nonalphanumeric characters (!, &, #, $, ^, <, >, -)
- Tokens or passwords must not contain a dictionary word or a slightly modified dictionary word.
- Tokens or passwords must not be the same as or similar to a recently used token.

### Applying authentication to an ElastiCache for Valkey and Redis OSS cluster

You can require that users enter a token (password) on a token-protected Valkey or Redis OSS server. To do this, include the parameter `--auth-token` (API: `AuthToken`) with the correct token when you create your replication group or cluster. Also include it in all subsequent commands to the replication group or cluster.

The following AWS CLI operation creates a replication group that has encryption in transit (TLS) enabled and required, with the **AUTH** token *`This-is-a-sample-token`*. Replace the subnet group `sng-test` with a subnet group that exists.

**Key parameters**

- **`--engine`** – Must be `valkey` or `redis`.
- **`--engine-version`** – If engine is Redis OSS, must be 4.0.10 or later.
- **`--transit-encryption-enabled`** – Required for authentication and HIPAA eligibility.
- **`--transit-encryption-mode`** – Must be `required` or omitted for authentication and HIPAA eligibility.
- **`--auth-token`** – Required for authentication when using AUTH tokens. For HIPAA eligibility, authentication is required (either via AUTH token or IAM authentication). This value must be the correct token for this token-protected Valkey or Redis OSS server.
- **`--cache-subnet-group`** – Required for HIPAA eligibility.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id authtestgroup \
    --replication-group-description authtest \
    --engine redis \
    --cache-node-type cache.m4.large \
    --num-node-groups 1 \
    --replicas-per-node-group 2 \
    --transit-encryption-enabled \
    --auth-token This-is-a-sample-token \
    --cache-subnet-group sng-test
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id authtestgroup ^
    --replication-group-description authtest ^
    --engine redis ^
    --cache-node-type cache.m4.large ^
    --num-node-groups 1 ^
    --replicas-per-node-group 2 ^
    --transit-encryption-enabled ^
```

```
    --auth-token This-is-a-sample-token ^
    --cache-subnet-group sng-test
```

### Modifying the AUTH token on an existing cluster

To make it easier to update your authentication, you can modify the **AUTH** token used on a cluster. You can make this modification if the engine version is Valkey 7.2 or higher or Redis 5.0.6 or higher. ElastiCache must also have encryption in transit enabled and transit encryption mode `required`. For more information, see ElastiCache in-transit encryption (TLS).

Modifying the auth token supports two strategies: ROTATE and SET. The ROTATE strategy adds an additional AUTH token to the server while retaining the previous token. The SET strategy updates the server to support just a single AUTH token. Make these modification calls with the `--apply-immediately` parameter to apply changes immediately.

### Rotating the AUTH token

To update a Valkey or Redis OSS server with a new **AUTH token**, call the `ModifyReplicationGroup` API with the `--auth-token` parameter as the new **AUTH** token and the `--auth-token-update-strategy` with the value ROTATE. After the ROTATE modification is complete, the cluster will support the previous AUTH token in addition to the one specified in the `auth-token` parameter. If no AUTH token was configured on the replication group before the AUTH token rotation, the cluster supports the AUTH token specified in the `--auth-token` parameter in addition to supporting connecting without authentication. See Setting the AUTH token to update the AUTH token to be required using update strategy SET.

> **Note**
>
> If you do not configure the AUTH token before, then once the modification is complete, the cluster will support no AUTH token in addition to the one specified in the auth-token parameter.

If this modification is performed on a server that already supports two AUTH tokens, the oldest AUTH token will also be removed during this operation. This allows a server to support up to two most recent AUTH tokens at a given time.

At this point, you can proceed by updating the client to use the latest AUTH token. After the clients are updated, you can use the SET strategy for **AUTH** token rotation (explained in the following section) to exclusively start using the new token.

The following AWS CLI operation modifies a replication group to rotate the **AUTH** token *This-is-the-rotated-token*.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
--replication-group-id authtestgroup \
--auth-token This-is-the-rotated-token \
--auth-token-update-strategy ROTATE \
--apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
--replication-group-id authtestgroup ^
--auth-token This-is-the-rotated-token ^
--auth-token-update-strategy ROTATE ^
--apply-immediately
```

**Setting the AUTH token**

To update a Valkey or Redis OSS server to support a single required **AUTH** token, call the `ModifyReplicationGroup` API operation with the `--auth-token` parameter with same value as the last AUTH token and the `--auth-token-update-strategy` parameter with the value SET. The SET strategy can only be used with a cluster that has 2 AUTH tokens or 1 optional AUTH token from using a ROTATE strategy before. After the modification is complete, the server supports only the AUTH token specified in the auth-token parameter.

The following AWS CLI operation modifies a replication group to set the AUTH token to *This-is-the-set-token*.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
--replication-group-id authtestgroup \
--auth-token This-is-the-set-token \
--auth-token-update-strategy SET \
--apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
--replication-group-id authtestgroup ^
--auth-token This-is-the-set-token ^
--auth-token-update-strategy SET ^
--apply-immediately
```

**Enabling authentication on an existing cluster**

To enable authentication on an existing Valkey or Redis OSS server, call the `ModifyReplicationGroup` API operation. Call `ModifyReplicationGroup` with the `--auth-token` parameter as the new token and the `--auth-token-update-strategy` with the value ROTATE.

After the ROTATE modification is complete, the cluster supports the **AUTH** token specified in the `--auth-token` parameter in addition to supporting connecting without authentication. Once all client applications are updated to authenticate to Valkey or Redis OSS with the AUTH token, use the SET strategy to mark the AUTH token as required. Enabling authentication is only supported on Valkey and Redis OSS servers with encryption in transit (TLS) enabled and required.

**Migrating from RBAC to AUTH**

If you are authenticating users with Valkey or Redis OSS Role-Based Access Control (RBAC) as described in Role-Based Access Control (RBAC), and you want to migrate to AUTH, use the following procedures. You can migrate using either console or CLI.

**To migrate from RBAC to AUTH using the console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the list in the upper-right corner, choose the AWS Region where the cluster that you want to modify is located.
3. In the navigation pane, choose the engine running on the cluster that you want to modify.

   A list of the chosen engine's clusters appears.
4. In the list of clusters, for the cluster that you want to modify, choose its name.
5. For **Actions**, choose **Modify**.

   The **Modify** window appears.

6. For **Access control**, choose **Valkey AUTH default user access** or **Redis OSS AUTH default user access**.
7. Under **Valkey AUTH token** or **Redis OSS AUTH token**, set a new token.
8. Choose **Preview changes** and then on the next screen, **Modify**.

**To migrate from RBAC to AUTH using the AWS CLI**

Use one of the following commands to configure a new optional **AUTH** token for your Valkey or Redis OSS replication group. Note that an optional Auth token will allow unauthenticated access to the replication group until the Auth token is marked as required, using the update strategy SET in the following step.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id test \
    --remove-user-groups \
    --auth-token This-is-a-sample-token \
    --auth-token-update-strategy ROTATE \
    --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id test ^
    --remove-user-groups ^
    --auth-token This-is-a-sample-token ^
    --auth-token-update-strategy ROTATE ^
    --apply-immediately
```

After executing the above command, you can update your Valkey or Redis OSS applications to authenticate to the ElastiCache replication group using the newly configured optional AUTH token. To complete the Auth token rotation, use the the update strategy SET in the subsequent command below. This will mark to the optional AUTH token as required. When the Auth token update completes, the replication group status will show as ACTIVE and all connections to this replication group will require authentication.

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
   --replication-group-id test \
   --auth-token This-is-a-sample-token \
   --auth-token-update-strategy SET \
   --apply-immediately
```

For Windows:

```
aws elasticache modify-replication-group ^
   --replication-group-id test ^
   --remove-user-groups ^
   --auth-token This-is-a-sample-token ^
   --auth-token-update-strategy SET ^
   --apply-immediately
```

For more information, see Authenticating with the Valkey and Redis OSS AUTH command.

> **Note**
>
> If you need to disable access control on an ElastiCache Cluster, see the section called "Disabling access control on an ElastiCache Valkey or Redis OSS cache".

### Migrating from password-based authentication (AUTH) to IAM authentication

This guide describes how to migrate your Amazon ElastiCache node-based cluster or serverless cache from password-based authentication (AUTH) to AWS Identity and Access Management (IAM) authentication without service interruption.

> **Note**
>
> We recommend testing this migration in a non-production environment before applying changes to your production environment.

#### Overview

IAM authentication provides enhanced security by eliminating the need for long-lived passwords. Instead, applications generate short-lived authentication tokens (valid for up to 15 minutes) using the AWS Signature Version 4 signing process.

### Prerequisites

Before you begin, verify the following:

- Your cache is running Redis OSS 7.0 or later, or Valkey. IAM authentication is not supported on earlier engine versions.
- In-transit encryption (TLS) is enabled on your cache. IAM authentication requires TLS. For more information, see the section called "In-transit encryption (TLS)".
- You have the necessary IAM permissions to create users and modify user groups. For more information, see the section called "How Amazon ElastiCache works with IAM".
- The IAM role or user that your application uses has an IAM policy that allows the `elasticache:Connect` action for the target cache and IAM user. For more information, see the section called "Authenticating with IAM".
- No additional IAM authentication limitations apply to your application. For the full list of limitations, see the section called "Authenticating with IAM".

### Migration process

The migration process involves creating both a password-based and an IAM-authenticated user, verifying connectivity, and then transitioning your applications to IAM authentication.

#### Step 1: Create users

Create two ElastiCache users so that your cache supports both password-based and IAM authentication during the migration period.

#### Create a password-based authentication user

> **Note**
>
> If you already have a password-based user configured, you can skip creating a new one and use your existing user.

Use the following AWS CLI command to create a password-based user.

For Linux, macOS, or Unix:

```
aws elasticache create-user \
  --user-id <user-id> \
  --user-name default \
  --engine <engine> \
  --passwords "<your-existing-auth-password>" \
  --access-string "on ~* +@all"
```

For Windows:

```
aws elasticache create-user ^
  --user-id <user-id> ^
  --user-name default ^
  --engine <engine> ^
  --passwords "<your-existing-auth-password>" ^
  --access-string "on ~* +@all"
```

Replace:

- *<user-id>* – A unique ID for this user.
- *<engine>* – The engine your cache uses: `valkey` or `redis`.
- *<your-existing-auth-password>* – The AUTH token currently configured on your cache.

**Create an IAM-authenticated user**

Use the following AWS CLI command to create an IAM-authenticated user.

For Linux, macOS, or Unix:

```
aws elasticache create-user \
  --user-id <iam-user-id> \
  --user-name <iam-user-name> \
  --authentication-mode Type=iam \
  --engine <engine> \
  --access-string "on ~* +@all"
```

For Windows:

```
aws elasticache create-user ^
  --user-id <iam-user-id> ^
```

```
  --user-name <iam-user-name> ^
  --authentication-mode Type=iam ^
  --engine <engine> ^
  --access-string "on ~* +@all"
```

Replace:

- *<iam-user-id>* – A unique ID for the IAM user.
- *<iam-user-name>* – The username for the IAM user.
- *<engine>* – The engine your cache uses: `valkey` or `redis`.

**Step 2: Create and attach user group**

If you already have a user group containing your password-based user, add the IAM-authenticated user to that existing group:

For Linux, macOS, or Unix:

```
aws elasticache modify-user-group \
  --user-group-id <user-group-id> \
  --user-ids-to-add <iam-user-id>
```

For Windows:

```
aws elasticache modify-user-group ^
  --user-group-id <user-group-id> ^
  --user-ids-to-add <iam-user-id>
```

Replace:

- *<user-group-id>* – The ID of your existing user group.
- *<iam-user-id>* – The user ID of the IAM-authenticated user created in Step 1.

Then skip to the section called “Step 3: Verify both authentication methods”.

Otherwise, create a new user group, add both users, and attach the group to your cache.

Use the following AWS CLI commands to create a user group with both users:

For Linux, macOS, or Unix:

```
aws elasticache create-user-group \
  --user-group-id <user-group-id> \
  --engine <engine> \
  --user-ids <user-id> <iam-user-id>
```

For Windows:

```
aws elasticache create-user-group ^
  --user-group-id <user-group-id> ^
  --engine <engine> ^
  --user-ids <user-id> <iam-user-id>
```

Replace:

- *<user-group-id>* – A unique ID for the user group.
- *<engine>* – The engine your cache uses: `valkey` or `redis`.
- *<user-id>*, *<iam-user-id>* – The user IDs created in Step 1.

Then modify your cache to use the new user group.

For node-based clusters:

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
  --replication-group-id <replication-group-id> \
  --auth-token-update-strategy DELETE \
  --user-group-ids-to-add <user-group-id>
```

For Windows:

```
aws elasticache modify-replication-group ^
  --replication-group-id <replication-group-id> ^
  --auth-token-update-strategy DELETE ^
  --user-group-ids-to-add <user-group-id>
```

Replace:

- *<replication-group-id>* – The ID of your replication group.
- *<user-group-id>* – The user group ID you created above.

For serverless caches:

For Linux, macOS, or Unix:

```
aws elasticache modify-serverless-cache \
  --serverless-cache-name <serverless-cache-name> \
  --user-group-id <user-group-id>
```

For Windows:

```
aws elasticache modify-serverless-cache ^
  --serverless-cache-name <serverless-cache-name> ^
  --user-group-id <user-group-id>
```

Replace:

- *<serverless-cache-name>* – The name of your serverless cache.
- *<user-group-id>* – The user group ID you created above.

**Step 3: Verify both authentication methods**

After completing Step 2, your cache supports both authentication methods. Verify that applications can connect using both methods before proceeding.

- Applications using password-based authentication can continue connecting with the password.
- Applications configured for IAM authentication can connect using IAM tokens.

**Step 4: Connect with IAM**

**Generate an IAM authentication token**

Generate a short-lived IAM authentication token using an AWS SigV4 pre-signed request. The following Python example demonstrates token generation.

```
import boto3
```

```
from botocore.auth import SigV4QueryAuth
from botocore.awsrequest import AWSRequest

cache_name = "<cache-name>"
user = "<username>"
region = "<region>"
expires = 900

session = boto3.Session()
credentials = session.get_credentials().get_frozen_credentials()

req = AWSRequest(
    method="GET",
    url=f"http://{cache_name}/",
    params={"Action": "connect", "User": user}
)
SigV4QueryAuth(credentials, "elasticache", region, expires=expires).add_auth(req)

token = req.url.replace("http://", "")
print(token)
```

> **Note**
>
> The generated token is valid for up to 15 minutes from creation.

### Connect using valkey-cli

Connect to your ElastiCache cache using the generated token. You can use either **valkey-cli** or **redis-cli** to connect to Valkey clusters. For Redis OSS clusters, use **redis-cli**.

```
valkey-cli -h <host> -p 6379 --tls
```

For cluster mode enabled clusters, add the `--cluster` flag:

```
valkey-cli -h <host> -p 6379 --tls --cluster
```

Then authenticate with the following command:

```
AUTH <username> <token>
```

Replace:

- *<host>* – The endpoint of your cache.
- *<username>* – The username of the IAM-authenticated user.
- *<token>* – The IAM authentication token you generated.

Expected output:

```
OK
```

Run the following command to validate the username:

```
ACL WHOAMI
```

**Step 5: Application integration**

For Java applications, use the default AWS Credentials provider chain to generate temporary security credentials. For more information, see the section called "Authenticating with IAM". For other languages, generate the IAM authentication token using the AWS Signature Version 4 signing process and pass it as the password in your client's AUTH command.

**Completing the migration**

After your applications can connect using IAM authentication, complete the following steps to finalize the migration.

**Step 1: Configure IAM authentication with fallback**

Before removing the password-based user, update your application code to use IAM authentication as the primary method while keeping the password-based user as a fallback.

In your application code:

- Configure your client to authenticate using IAM-generated tokens as the primary method.
- Add a fallback mechanism so that if IAM authentication fails (for example, due to token expiry or a generation error), the client retries using the password-based AUTH credential.
- Log all authentication attempts, including fallbacks and retries, so you can monitor whether any connections are falling back to password-based auth.

**Step 2: Monitor and verify**

Review your application logs and the `IamAuthenticationExpirations` and `IamAuthenticationThrottling` Amazon CloudWatch metrics over at least 24–48 hours to confirm all connections are through IAM.

A non-zero value for either metric warrants investigation:

- `IamAuthenticationExpirations` – An IAM authenticated connection is automatically disconnected after 12 hours. The connection can be prolonged by sending an `AUTH` or `HELLO` command with a new IAM authentication token.
- `IamAuthenticationThrottling` – Indicates too many authentication requests, which might mean your application is generating tokens too aggressively or has connection pooling issues.

For node-based clusters, you can perform additional engine-level checks:

- `ACL LOG` – Check for failed authentication attempts. Look for entries with `reason=auth`.
- `CLIENT LIST` – Verify connected users. Check the `user=` field to confirm clients are using the IAM user.

`ACL LOG` and `CLIENT LIST` are available on node-based clusters only. For ElastiCache Serverless deployments, rely on application-side logging and Amazon CloudWatch metrics.

**Step 3: Remove the password-based user**

Once you confirm that all applications use IAM authentication, remove the password-based user from the user group.

**Valkey caches**

Remove the password-based user directly:

For Linux, macOS, or Unix:

```
aws elasticache modify-user-group \
   --user-group-id <user-group-id> \
   --user-ids-to-remove <password-user-id>
```

For Windows:

```
aws elasticache modify-user-group ^
   --user-group-id <user-group-id> ^
   --user-ids-to-remove <password-user-id>
```

**Redis OSS caches**

For Redis OSS caches, the user group must always contain a user with the username `default`. If the password-based user you created in Step 1 has the username `default`, you must create a disabled placeholder user to replace it before removing it:

For Linux, macOS, or Unix:

```
aws elasticache create-user \
   --user-id <disabled-default-id> \
   --user-name default \
   --engine redis \
   --no-password-required \
   --access-string "off ~* -@all"
```

For Windows:

```
aws elasticache create-user ^
   --user-id <disabled-default-id> ^
   --user-name default ^
   --engine redis ^
   --no-password-required ^
   --access-string "off ~* -@all"
```

Then add the disabled user to the group and remove the password-based user:

For Linux, macOS, or Unix:

```
aws elasticache modify-user-group \
   --user-group-id <user-group-id> \
   --user-ids-to-add <disabled-default-id> \
   --user-ids-to-remove <password-user-id>
```

For Windows:

```
aws elasticache modify-user-group ^
   --user-group-id <user-group-id> ^
```

```
    --user-ids-to-add <disabled-default-id> ^
    --user-ids-to-remove <password-user-id>
```

If the password-based user you created in Step 1 does not have the username `default`, you can remove it directly using the same command shown for Valkey caches above.

Replace:

- *<disabled-default-id>* – A unique ID for the disabled placeholder user (Redis OSS only).
- *<user-group-id>* – The ID of the user group associated with your cache.
- *<password-user-id>* – The user ID of the password-based user to remove.

> **Note**
>
> Do not delete the password-based user after removing it from the user group. Keep it available in case you need to roll back to password-based authentication.

**Step 4: Confirm post-removal**

After removing the password-based user from the user group, monitor the `AuthenticationFailures` Amazon CloudWatch metric. A sustained value of zero confirms no authentication failures occurring, including any residual password-based attempts. Set a CloudWatch alarm on this metric to detect unexpected attempts.

For node-based clusters, you can also verify using `ACL LOG` and `CLIENT LIST`. For serverless caches, rely on application-side logging and CloudWatch metrics.

**Rollback procedure**

Re-add the password-based user to the user group, then restore password-based authentication in your application while you investigate.

For Valkey caches:

For Linux, macOS, or Unix:

```
aws elasticache modify-user-group \
    --user-group-id <user-group-id> \
    --user-ids-to-add <password-user-id>
```

For Windows:

```
aws elasticache modify-user-group ^
   --user-group-id <user-group-id> ^
   --user-ids-to-add <password-user-id>
```

For Redis OSS caches where the password-based user does not have the username `default`, use the same command above.

For Redis OSS caches where you created a disabled placeholder user in the section called "Step 3: Remove the password-based user", add the password-based user and remove the disabled placeholder user in a single operation:

For Linux, macOS, or Unix:

```
aws elasticache modify-user-group \
   --user-group-id <user-group-id> \
   --user-ids-to-add <password-user-id> \
   --user-ids-to-remove <disabled-default-id>
```

For Windows:

```
aws elasticache modify-user-group ^
   --user-group-id <user-group-id> ^
   --user-ids-to-add <password-user-id> ^
   --user-ids-to-remove <disabled-default-id>
```

Replace:

- *<user-group-id>* – The ID of the user group associated with your cache.
- *<password-user-id>* – The user ID of the password-based user to re-add.
- *<disabled-default-id>* – The user ID of the disabled placeholder user (Redis OSS only).

### Disabling access control on an ElastiCache Valkey or Redis OSS cache

Follow the instructions below to disable access control on a Valkey or Redis OSS TLS-enabled cache. Your cache will have one of two different types of configurations: AUTH default user access or User group access control list (RBAC). If your cache was created with the AUTH configuration, you have to change it to the RBAC configuration before you can disable the cache by removing

the user groups. If your cache was created with the RBAC configuration, you can go straight into disabling it.

> **User group required for public endpoints**
>
> ElastiCache Serverless caches with a public endpoint require a user group with IAM-authenticated users. You cannot remove the user group from a cache with a public endpoint.

**To disable a Valkey or Redis OSS serverless cache configured with RBAC**

1. Remove the user groups to disable the access control.

```
aws elasticache modify-serverless-cache --serverless-cache-name <serverless-cache>
 --remove-user-group
```

2. (Optional) Verify that no user groups are associated with the serverless cache.

```
aws elasticache describe-serverless-caches --serverless-cache-name <serverless-
cache>
{
    "..."
    "UserGroupId": ""
    "..."
}
```

**To disable a Valkey or Redis OSS cache with configured with an AUTH token**

1. Change the AUTH token to RBAC and specify a user group to add.

```
aws elasticache modify-replication-group --replication-group-id <replication-group-
id-value> --auth-token-update-strategy DELETE --user-group-ids-to-add <user-group-
value>
```

2. Verify that the AUTH token got disabled and that a user group was added.

```
aws elasticache describe-replication-groups --replication-group-id <replication-
group-id-value>
{
```

```
    "..."
    "AuthTokenEnabled": false,
    "UserGroupIds": [
        "<user-group-value>"
    ]
    "..."
}
```

3. Remove the user groups to disable the access control.

```
aws elasticache modify-replication-group --replication-group-id <replication-group-
value> --user-group-ids-to-remove <user-group-value>
{
    "..."
    "PendingModifiedValues": {
    "UserGroups": {
      "UserGroupIdsToAdd": [],
      "UserGroupIdsToRemove": [
        "<user-group-value>"
      ]
    }
    "..."
}
```

4. (Optional) Verify that no user groups are associated with the cluster. The `AuthTokenEnabled` field should also read false.

```
aws elasticache describe-replication-groups --replication-group-id <replication-
group-value>
"AuthTokenEnabled": false
```

**To disable a Valkey or Redis OSS cluster configured with RBAC**

1. Remove the user groups to disable the access control.

```
aws elasticache modify-replication-group --replication-group-id <replication-group-
value> --user-group-ids-to-remove <user-group-value>
{
    "..."
    "PendingModifiedValues": {
    "UserGroups": {
      "UserGroupIdsToAdd": [],
```

```
      "UserGroupIdsToRemove": [
        "<user-group-value>"
      ]
    }
    "..."
}
```

2. (Optional) Verify that no user groups are associated with the cluster. The `AuthTokenEnabled` field should also read false.

```
aws elasticache describe-replication-groups --replication-group-id <replication-
group-value>
"AuthTokenEnabled": false
```

# Internetwork traffic privacy

Amazon ElastiCache uses the following techniques to secure your cache data and protect it from unauthorized access:

- **Amazon VPCs and ElastiCache security** explains the type of security group you need for your installation.
- **Identity and Access Management for Amazon ElastiCache** for granting and limiting actions of users, groups, and roles.

For ElastiCache Serverless caches with a public endpoint, connections are secured with IAM authentication and TLS 1.3 instead of Amazon Virtual Private Cloud (Amazon VPC) security groups. All users in the cache's user group must use IAM authentication.

**Topics**

- Amazon VPCs and ElastiCache security
- ElastiCache API and interface VPC endpoints (AWS PrivateLink)
- Subnets and subnet groups

# Amazon VPCs and ElastiCache security

Because data security is important, ElastiCache provides means for you to control who has access to your data. How you control access to your data is dependent upon whether or not you launched your clusters in an Amazon Virtual Private Cloud (Amazon VPC) or Amazon EC2-Classic.

> **Important**
>
> We have deprecated the use of Amazon EC2-Classic for launching ElastiCache clusters. All node-based clusters and ElastiCache Serverless caches with a VPC endpoint are launched in Amazon Virtual Private Cloud. ElastiCache Serverless caches can also be configured with a public endpoint, which does not require VPC configuration.

The Amazon Virtual Private Cloud (Amazon VPC) service defines a virtual network that closely resembles a traditional data center. When you configure your Amazon VPC you can select its IP address range, create subnets, and configure route tables, network gateways, and security settings. You can also add a cluster to the virtual network, and control access to the cluster by using Amazon VPC security groups.

This section explains how to manually configure an ElastiCache cluster in an Amazon VPC. This information is intended for users who want a deeper understanding of how ElastiCache and Amazon VPC work together.

**Topics**


- Understanding ElastiCache and Amazon VPCs
- Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC
- Creating a Virtual Private Cloud (VPC)
- Connecting to a cache running in an Amazon VPC

## Understanding ElastiCache and Amazon VPCs

ElastiCache is fully integrated with the Amazon Virtual Private Cloud (Amazon VPC). For ElastiCache users, this means the following:

- If your AWS account supports only the EC2-VPC platform, ElastiCache always launches your cluster in an Amazon VPC.
- If you're new to AWS, your clusters will be deployed into an Amazon VPC. A default VPC will be created for you automatically.
- If you have a default VPC and don't specify a subnet when you launch a cluster, the cluster launches into your default Amazon VPC.

For more information, see Detecting Your Supported Platforms and Whether You Have a Default VPC.

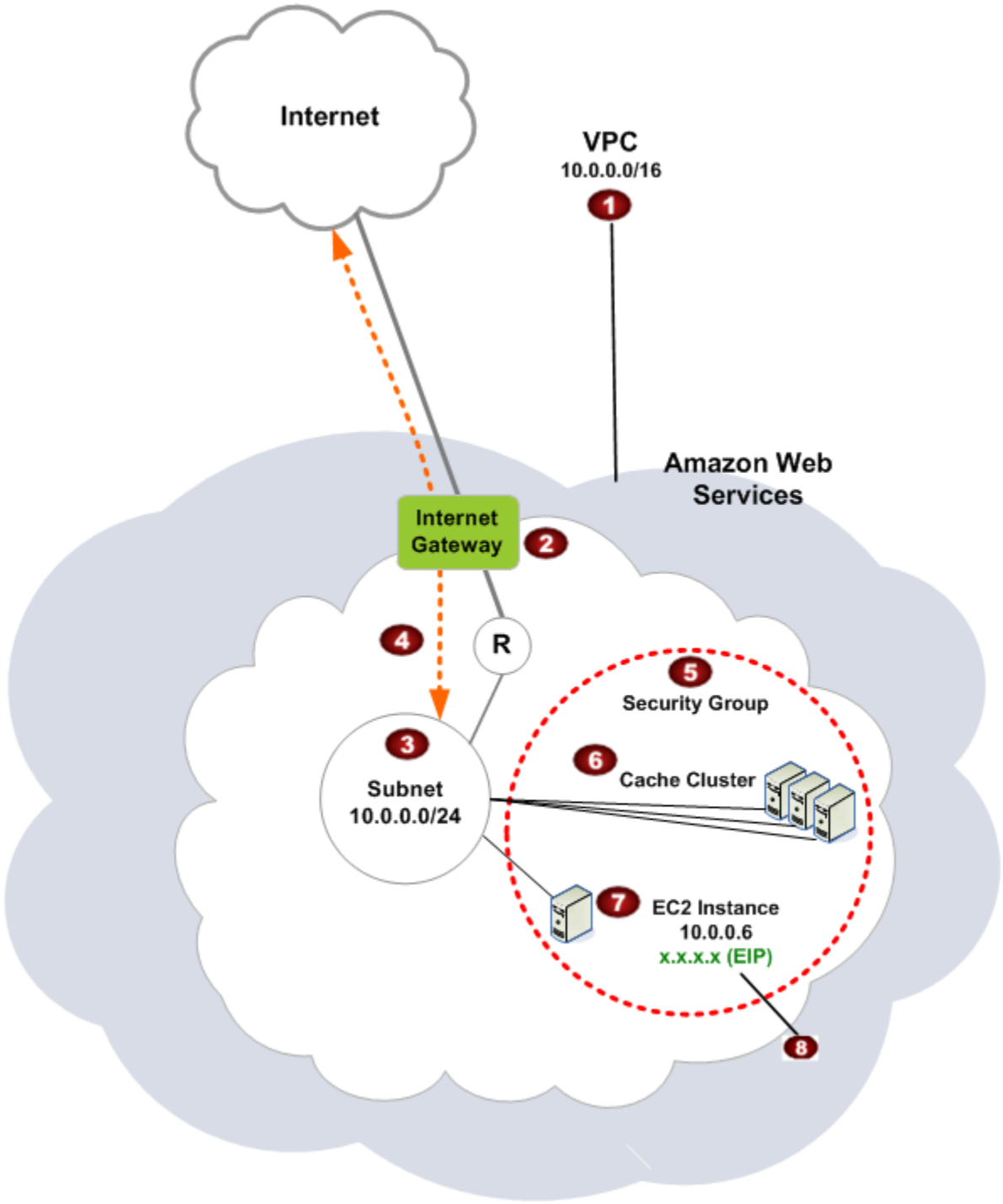
With Amazon Virtual Private Cloud, you can create a virtual network in the AWS cloud that closely resembles a traditional data center. You can configure your Amazon VPC, including selecting its IP address range, creating subnets, and configuring route tables, network gateways, and security settings.

The basic functionality of ElastiCache is the same in a virtual private cloud; ElastiCache manages software upgrades, patching, failure detection and recovery whether your clusters are deployed inside or outside an Amazon VPC.

ElastiCache cache nodes deployed outside an Amazon VPC are assigned an IP address to which the endpoint/DNS name resolves. This provides connectivity from Amazon Elastic Compute Cloud (Amazon EC2) instances. When you launch an ElastiCache cluster into an Amazon VPC private subnet, every cache node is assigned a private IP address within that subnet.

### Overview of ElastiCache in an Amazon VPC

The following diagram and table describe the Amazon VPC environment, along with ElastiCache clusters and Amazon EC2 instances that are launched in the Amazon VPC.




| | |
|---|---|
| 1 | The Amazon VPC is an isolated portion of the AWS Cloud that is assigned its own block of IP addresses. |
| 2 | |

| | |
|---|---|
| | An Internet gateway connects your Amazon VPC directly to the Internet and provides access to other AWS resources such as Amazon Simple Storage Service (Amazon S3) that are running outside your Amazon VPC. |
| 3 | An Amazon VPC subnet is a segment of the IP address range of an Amazon VPC where you can isolate AWS resources according to your security and operational needs. |
| 4 | A routing table in the Amazon VPC directs network traffic between the subnet and the Internet. The Amazon VPC has an implied router, which is symbolized in this diagram by the circle with the R. |
| 5 | An Amazon VPC security group controls inbound and outbound traffic for your ElastiCac he clusters and Amazon EC2 instances. |
| 6 | You can launch an ElastiCache cluster in the subnet. The cache nodes have private IP addresses from the subnet's range of addresses. |
| 7 | You can also launch Amazon EC2 instances in the subnet. Each Amazon EC2 instance has a private IP address from the subnet's range of addresses. The Amazon EC2 instance can connect to any cache node in the same subnet. |
| 8 | For an Amazon EC2 instance in your Amazon VPC to be reachable from the Internet, you need to assign a static, public address called an Elastic IP address to the instance. |

**Prerequisites**

To create an ElastiCache cluster within an Amazon VPC, your Amazon VPC must meet the following requirements:

- The Amazon VPC must allow nondedicated Amazon EC2 instances. You cannot use ElastiCache in an Amazon VPC that is configured for dedicated instance tenancy.
- A cache subnet group must be defined for your Amazon VPC. ElastiCache uses that cache subnet group to select a subnet and IP addresses within that subnet to associate with your VPC endpoints or cache nodes.

- CIDR blocks for each subnet must be large enough to provide spare IP addresses for ElastiCache to use during maintenance activities.

### Routing and security

You can configure routing in your Amazon VPC to control where traffic flows (for example, to the Internet gateway or virtual private gateway). With an Internet gateway, your Amazon VPC has direct access to other AWS resources that are not running in your Amazon VPC. If you choose to have only a virtual private gateway with a connection to your organization's local network, you can route your Internet-bound traffic over the VPN and use local security policies and firewall to control egress. In that case, you incur additional bandwidth charges when you access AWS resources over the Internet.

You can use Amazon VPC security groups to help secure the ElastiCache clusters and Amazon EC2 instances in your Amazon VPC. Security groups act like a firewall at the instance level, not the subnet level.

> **Note**
>
> We strongly recommend that you use DNS names to connect to your cache nodes, as the underlying IP address can change.

### Amazon VPC documentation

Amazon VPC has its own set of documentation to describe how to create and use your Amazon VPC. The following table gives links to the Amazon VPC guides.

| Description | Documentation |
| --- | --- |
| How to get started using Amazon VPC | Getting started with Amazon VPC |
| How to use Amazon VPC through the AWS Management Console | Amazon VPC User Guide |
| Complete descriptions of all the Amazon VPC commands | Amazon EC2 Command Line Reference (the Amazon VPC commands are found in the Amazon EC2 reference) |

| Description | Documentation |
| --- | --- |
| Complete descriptions of the Amazon VPC API operations, data types, and errors | Amazon EC2 Command Line Reference (the Amazon VPC API operations are found in the Amazon EC2 reference) |
| Information for the network administrator who needs to configure the gateway at your end of an optional IPsec VPN connection | What is AWS Site-to-Site VPN? |

For more detailed information about Amazon Virtual Private Cloud, see Amazon Virtual Private Cloud.

# Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC

Amazon ElastiCache supports the following scenarios for accessing a cache in an Amazon VPC:

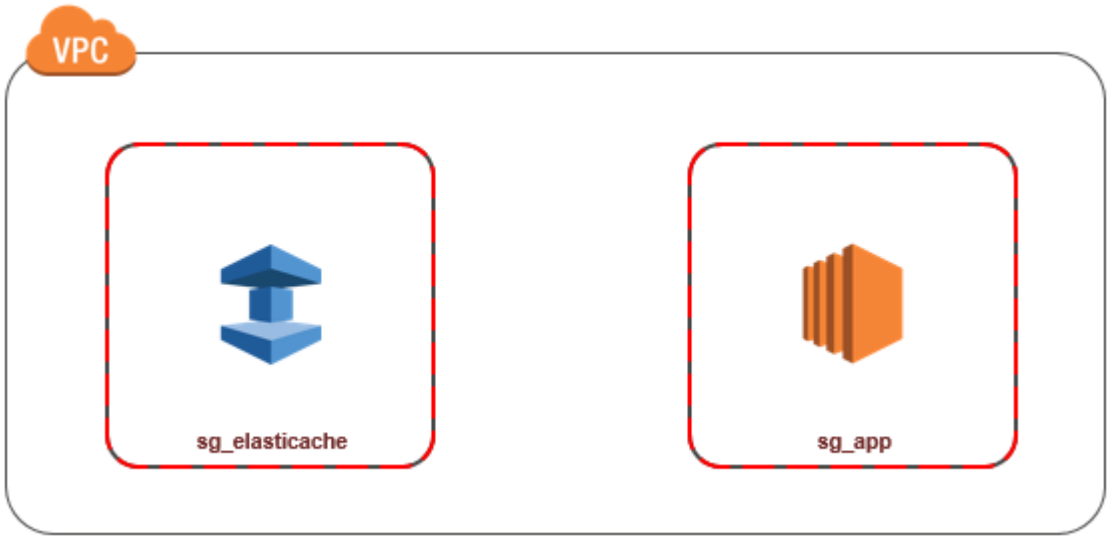
**Contents**


- Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in the Same Amazon VPC
- Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs
  - Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs in the Same Region
    - Using Transit Gateway
  - Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs in Different Regions
    - Using Transit VPC
- Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center
  - Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center Using VPN Connectivity
  - Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center Using Direct Connect


## Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in the Same Amazon VPC

The most common use case is when an application deployed on an EC2 instance needs to connect to a cache in the same VPC.

The following diagram illustrates this scenario.




The simplest way to manage access between EC2 instances and caches in the same VPC is to do the following:

1. Create a VPC security group for your cache. This security group can be used to restrict access to the cache. For example, you can create a custom rule for this security group that allows TCP access using the port you assigned to the cache when you created it and an IP address you will use to access the cache.

   The default port for Memcached caches is 11211.

   The default port for Valkey and Redis OSS caches is 6379.

2. Create a VPC security group for your EC2 instances (web and application servers). This security group can, if needed, allow access to the EC2 instance from the Internet via the VPC's routing table. For example, you can set rules on this security group to allow TCP access to the EC2 instance over port 22.
3. Create custom rules in the security group for your cache that allow connections from the security group you created for your EC2 instances. This would allow any member of the security group to access the caches.

**Note**

If you are planning to use Local Zones, ensure that you have enabled them. When you create a subnet group in that local zone, your VPC is extended to that Local Zone and your

VPC will treat the subnet as any subnet in any other Availability Zone. All relevant gateways and route tables will be automatically adjusted.

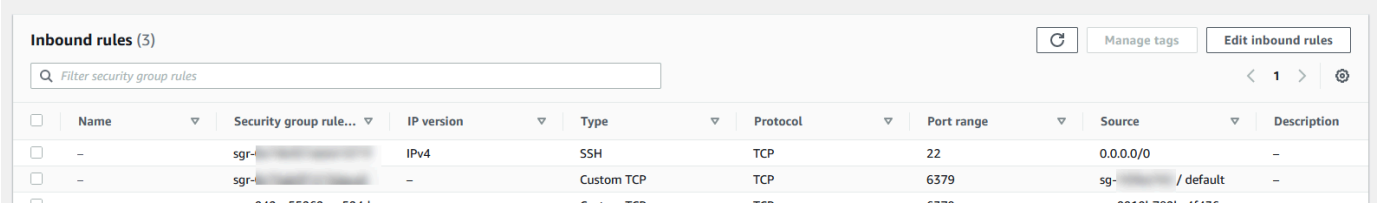
**To create a rule in a VPC security group that allows connections from another security group**

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc.
2. In the navigation pane, choose **Security Groups**.
3. Select or create a security group that you will use for your cache. Under **Inbound Rules**, select **Edit Inbound Rules** and then select **Add Rule**. This security group will allow access to members of another security group.
4. From **Type** choose **Custom TCP Rule**.
   a. For **Port Range**, specify the port you used when you created your cache.

      The default port for Memcached caches is 11211.

      The default port for Valkey and Redis OSS caches and replication groups is 6379.

   b. In the **Source** box, start typing the ID of the security group. From the list select the security group you will use for your Amazon EC2 instances.
5. Choose **Save** when you finish.




## Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs

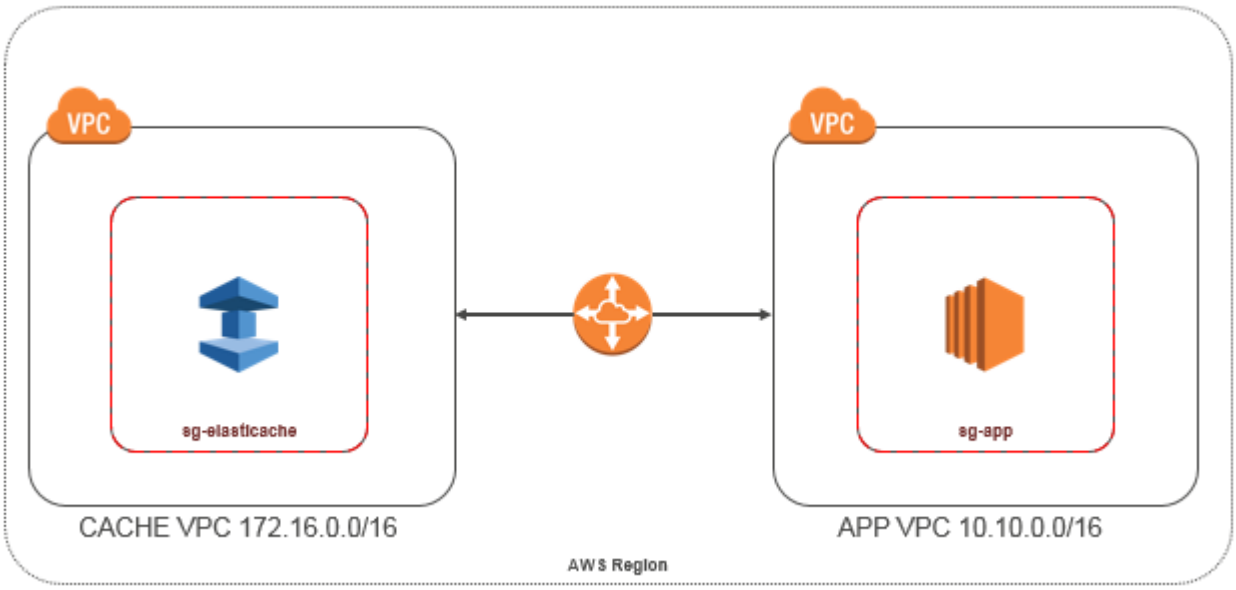
When your cache is in a different VPC from the EC2 instance you are using to access it, there are several ways to access the cache. If the cache and EC2 instance are in different VPCs but in the same Region, you can use VPC peering. If the cache and the EC2 instance are in different Regions, you can use inter-Region VPC peering or create VPN connectivity between Regions.

### Topics

- Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs in the Same Region
- Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs in Different Regions

**Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs in the Same Region**

The following diagram illustrates accessing a cache by an Amazon EC2 instance in a different Amazon VPC in the same region using an Amazon VPC peering connection.




*Cache accessed by an Amazon EC2 instance in a different Amazon VPC within the same Region - VPC Peering Connection*

A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses. Instances in either VPC can communicate with each other as if they are within the same network. You can create a VPC peering connection between your own Amazon VPCs, or with an Amazon VPC in another AWS account within a single region. To learn more about Amazon VPC peering, see the VPC documentation.

> **Note**
>
> DNS name resolution may fail for peered VPCs, depending on the configurations applied to the ElastiCache VPC. To resolve this, both VPCs must be enabled for DNS hostnames and DNS resolution. For more information, see Enable DNS resolution for a VPC peering connection.

**To access a cache in a different Amazon VPC over peering**

1. Make sure that the two VPCs do not have an overlapping IP range or you will not be able to peer them.
2. Peer the two VPCs. For more information, see Creating and Accepting an Amazon VPC Peering Connection.
3. Update your routing table. For more information, see Updating Your Route Tables for a VPC Peering Connection

   Following is what the route tables look like for the example in the preceeding diagram. Note that **pcx-a894f1c1** is the peering connection.

| Destination | Target |
|---|---|
| 172.16.0.0/16 | local |
| 10.10.0.0/16 | pcx-a894f1c1 |

| Destination | Target |
|---|---|
| 10.10.0.0/16 | local |
| 0.0.0.0/0 | igw-bfdcccd8 |
| 172.16.0.0/16 | pcx-a894f1c1 |

   *VPC Routing Table*
4. Modify the Security Group of your ElastiCache cache to allow inbound connection from the Application security group in the peered VPC. For more information, see Reference Peer VPC Security Groups.

Accessing a cache over a peering connection will incur additional data transfer costs.

**Using Transit Gateway**

A transit gateway enables you to attach VPCs and VPN connections in the same AWS Region and route traffic between them. A transit gateway works across AWS accounts, and you can use AWS

Resource Access Manager to share your transit gateway with other accounts. After you share a transit gateway with another AWS account, the account owner can attach their VPCs to your transit gateway. A user from either account can delete the attachment at any time.

You can also use a transit gateway to connect a cache to an application in a different Amazon VPC. Attach the Amazon VPC that contains the cache and the Amazon VPC that contains the application to the same transit gateway. Then update the route tables of both VPCs to route traffic between them.

You must also allow the application to reach the cache. Modify the security group of the cache to allow inbound TCP traffic from the security group or IP address range of the application. Use the port that the cache uses.

You can also create a peering connection attachment between transit gateways in different AWS Regions. This enables you to route traffic between the transit gateways' attachments across different Regions.

For more information, see Transit gateways.

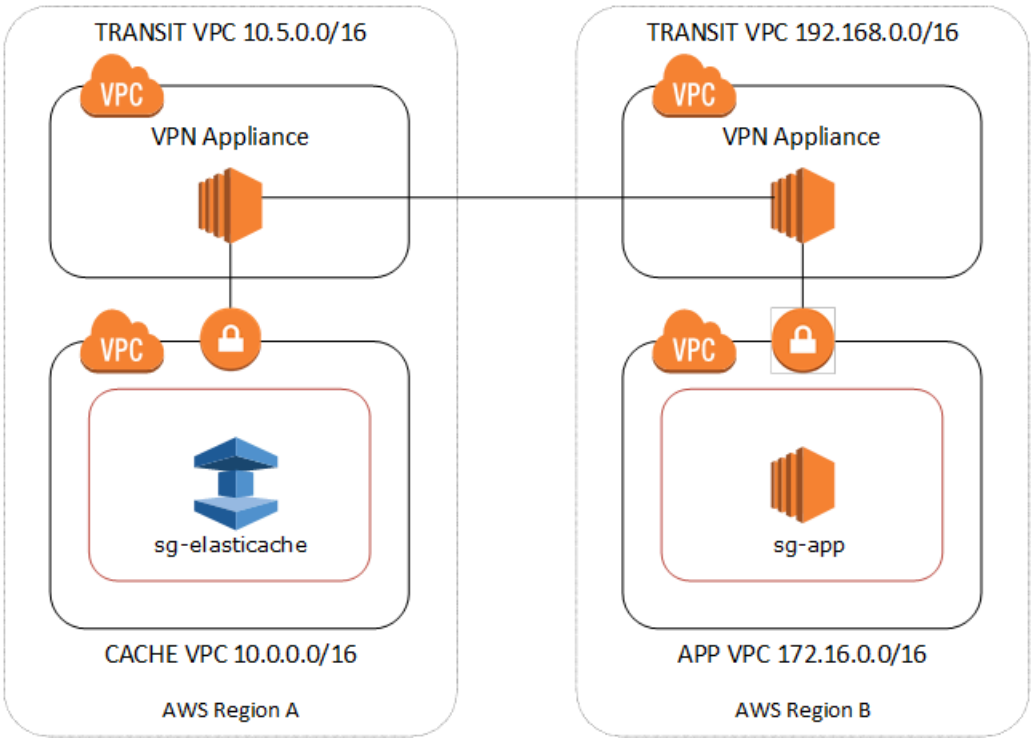
**Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in Different Amazon VPCs in Different Regions**

If your cache and Amazon EC2 instance are in different Regions, you can connect them by using an inter-Region Amazon VPC peering connection. Resources in VPCs in different AWS Regions can then communicate using private IP addresses, without a gateway, VPN connection, or separate network appliance. Inter-Region Amazon VPC peering is available in all commercial AWS Regions. For more information, see the Amazon VPC Peering Guide.

Alternatively, you can use a transit VPC, as described in the following section.

**Using Transit VPC**

An alternative to using VPC peering, another common strategy for connecting multiple, geographically disperse VPCs and remote networks is to create a transit VPC that serves as a global network transit center. A transit VPC simplifies network management and minimizes the number of connections required to connect multiple VPCs and remote networks. This design can save time and effort and also reduce costs, as it is implemented virtually without the traditional expense of establishing a physical presence in a colocation transit hub or deploying physical network gear.




*Connecting across different VPCs in different regions*

Once the Transit Amazon VPC is established, an application deployed in a “spoke” VPC in one region can connect to an ElastiCache cache in a “spoke” VPC within another region.

**To access a cache in a different VPC within a different AWS Region**

1. Deploy a Transit VPC Solution. For more information, see, AWS Transit Gateway.
2. Update the VPC routing tables in the App and Cache VPCs to route traffic through the VGW (Virtual Private Gateway) and the VPN Appliance. In case of Dynamic Routing with Border Gateway Protocol (BGP) your routes may be automatically propagated.
3. Modify the Security Group of your ElastiCache cache to allow inbound connection from the Application instances IP range. Note that you will not be able to reference the application server Security Group in this scenario.

Accessing a cache across regions will introduce networking latencies and additional cross-region data transfer costs.

### Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center

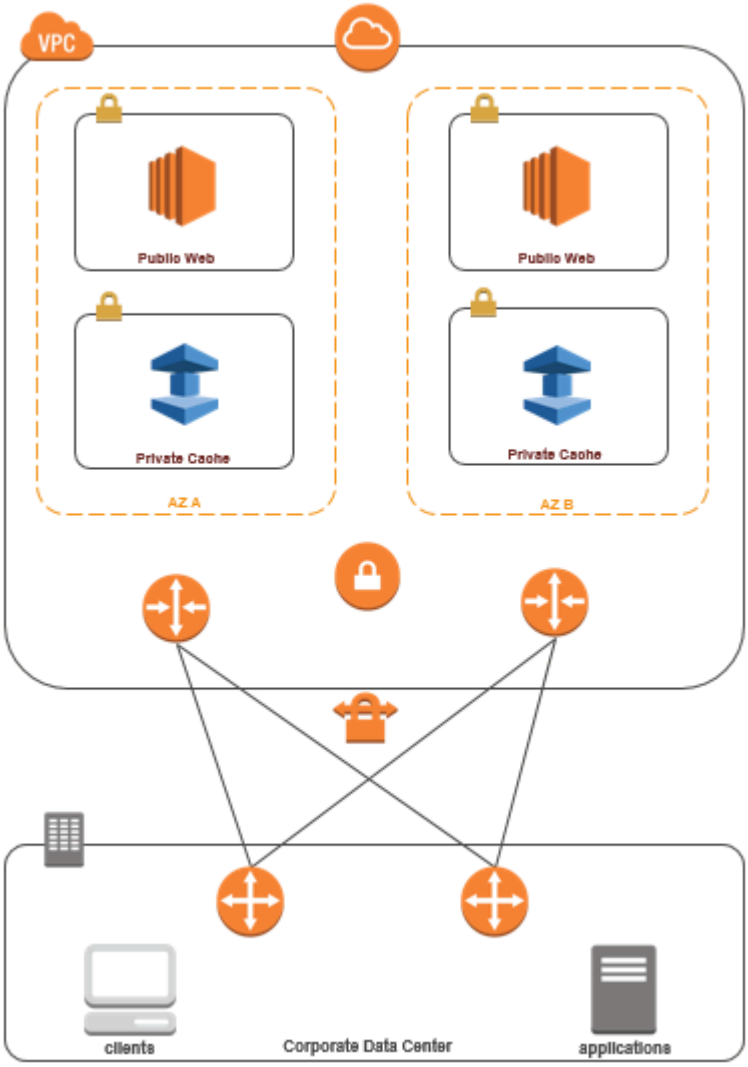
Another possible scenario is a Hybrid architecture where clients or applications in the customer's data center may need to access an ElastiCache cache in the VPC. This scenario is also supported providing there is connectivity between the customers' VPC and the data center either through VPN or Direct Connect.

**Topics**

- Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center Using VPN Connectivity
- Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center Using Direct Connect

### Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center Using VPN Connectivity

The following diagram illustrates accessing an ElastiCache cache from an application running in your corporate network using VPN connections.




*Connecting to ElastiCache from your data center via a VPN*

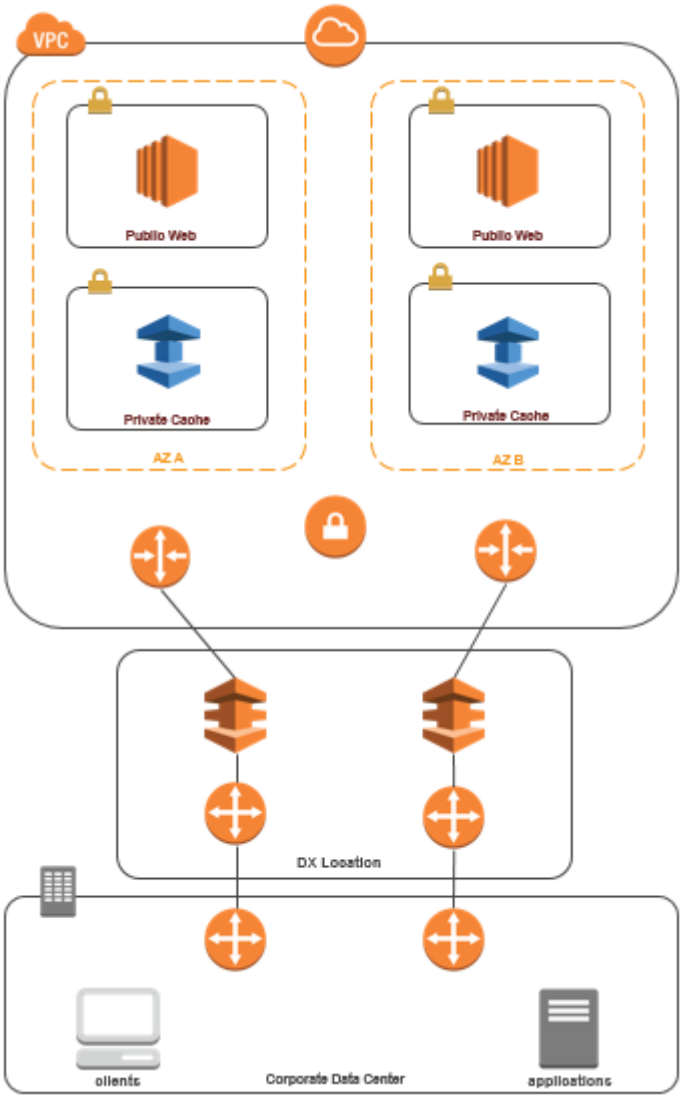
**To access a cache in a VPC from on-prem application over VPN connection**

1. Establish VPN Connectivity by adding a hardware Virtual Private Gateway to your VPC. For more information, see Adding a Hardware Virtual Private Gateway to Your VPC.
2. Update the VPC routing table for the subnet where your ElastiCache cache is deployed to allow traffic from your on-premises application server. In case of Dynamic Routing with BGP your routes may be automatically propagated.
3. Modify the Security Group of your ElastiCache cache to allow inbound connection from the on-premises application servers.

Accessing a cache over a VPN connection will introduce networking latencies and additional data transfer costs.

### Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center Using Direct Connect

The following diagram illustrates accessing an ElastiCache cache from an application running on your corporate network using Direct Connect.




*Connecting to ElastiCache from your data center via Direct Connect*

**To access an ElastiCache cache from an application running in your network using Direct Connect**

1. Establish Direct Connect connectivity. For more information, see, Getting Started with AWS Direct Connect.
2. Modify the Security Group of your ElastiCache cache to allow inbound connection from the on-premises application servers.

Accessing a cache over DX connection may introduce networking latencies and additional data transfer charges.

## Creating a Virtual Private Cloud (VPC)

In this example, you create an Amazon VPC with a private subnet for each Availability Zone.

### Creating an Amazon VPC (Console)

1. Sign in to the AWS Management Console, and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the VPC dashboard, choose **Create VPC**.
3. Under **Resources** to create, choose **VPC and more**.
4. Under **Number of Availability Zones (AZs)**, choose the number of Availability Zones you want to launch your subnets in.
5. Under **Number of public subnets**, choose the number of public subnets you want to add to your VPC.
6. Under **Number of private subnets**, choose the number of private subnets you want to add to your VPC.

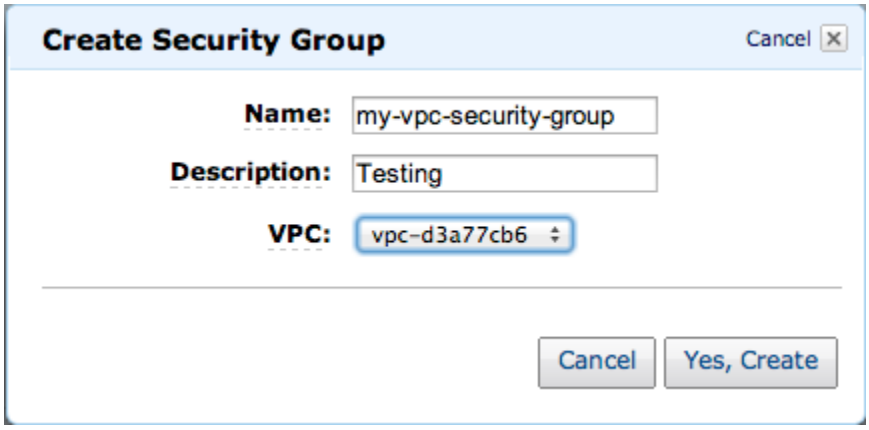
   > **Tip**
   >
   > Make a note of your subnet identifiers, and which are public and private. You will need this information later when you launch your clusters and add an Amazon EC2 instance to your Amazon VPC.

7. Create an Amazon VPC security group. You will use this group for your cluster and your Amazon EC2 instance.

   a. In the navigation pane of the Amazon VPC Management console, choose **Security Groups**.

   b. Choose **Create Security Group**.

   c. Type a name and a description for your security group in the corresponding boxes. In the **VPC** box, choose the identifier for your Amazon VPC.




d. When the settings are as you want them, choose **Yes, Create**.

8. Define a network ingress rule for your security group. This rule will allow you to connect to your Amazon EC2 instance using Secure Shell (SSH).

   a. In the navigation list, choose **Security Groups**.

   b. Find your security group in the list, and then choose it.

   c. Under **Security Group**, choose the **Inbound** tab. In the **Create a new rule** box, choose **SSH**, and then choose **Add Rule**.

   d. Set the following values for your new inbound rule to allow HTTP access:

      - Type: HTTP
      - Source: 0.0.0.0/0

      Choose **Apply Rule Changes**.

Now you are ready to create a cache subnet group and launch a cluster in your Amazon VPC.

- Creating a subnet group
- Creating a Memcached cluster (console).
- Creating a Valkey (cluster mode disabled) cluster (Console).

## Connecting to a cache running in an Amazon VPC

This example shows how to launch an Amazon EC2 instance in your Amazon VPC. You can then log in to this instance and access the ElastiCache cache that is running in the Amazon VPC.

### Connecting to a cache running in an Amazon VPC (Console)

In this example, you create an Amazon EC2 instance in your Amazon VPC. You can use this Amazon EC2 instance to connect to cache nodes running in the Amazon VPC.

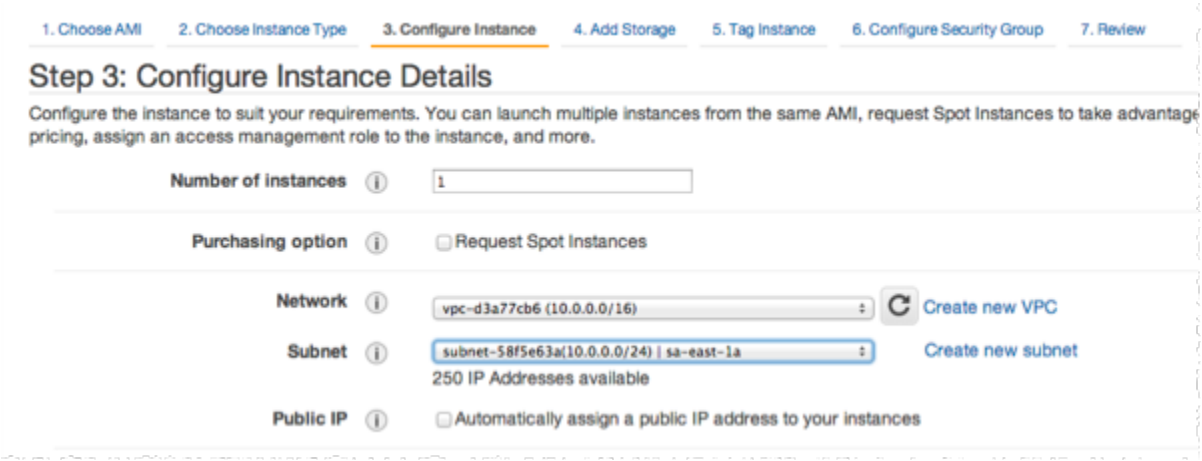
> **Note**
>
> For information about using Amazon EC2, see the Amazon EC2 Getting Started Guide in the Amazon EC2 documentation.

#### To create an Amazon EC2 instance in your Amazon VPC using the Amazon EC2 console

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the console, choose **Launch Instance** and follow these steps:
3. On the **Choose an Amazon Machine Image (AMI)** page, choose the 64-bit Amazon Linux AMI, and then choose **Select**.
4. On the **Choose an Instance Type** page, choose **3. Configure Instance**.
5. On the **Configure Instance Details** page, make the following selections:

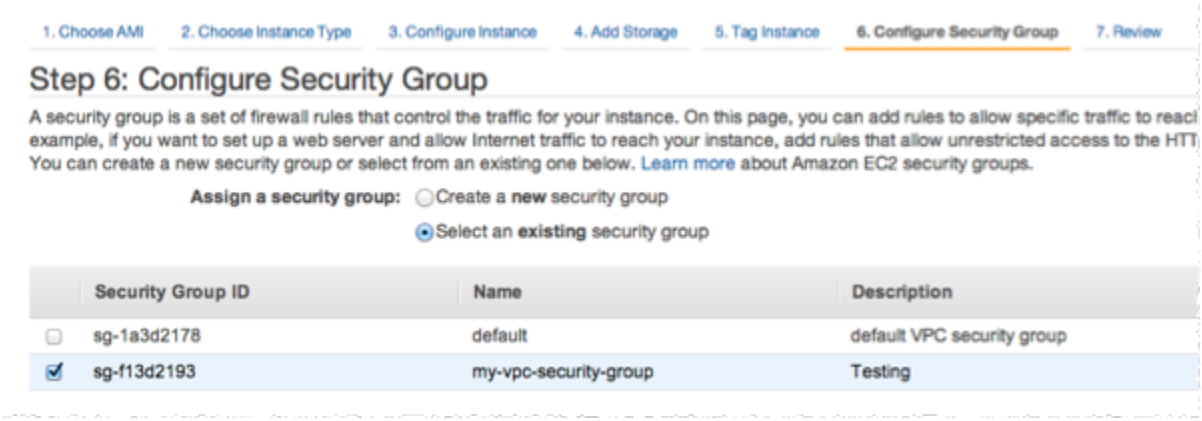
    a. In the **Network** list, choose your Amazon VPC.

    b. In the **Subnet** list, choose your public subnet.



When the settings are as you want them, choose **4. Add Storage**.

6. On the **Add Storage** page, choose **5. Tag Instance**.
7. On the **Tag Instance** page, type a name for your Amazon EC2 instance, and then choose **6. Configure Security Group**.
8. On the **Configure Security Group** page, choose **Select an existing security group**. For more information on security groups, see Amazon EC2 security groups for Linux instances.




Choose the name of your Amazon VPC security group, and then choose **Review and Launch**.

9. On the **Review Instance and Launch** page, choose **Launch**.
10. In the **Select an existing key pair or create a new key pair** window, specify a key pair that you want to use with this instance.

> **Note**
>
> For information about managing key pairs, see the Amazon EC2 Getting Started Guide.

11. When you are ready to launch your Amazon EC2 instance, choose **Launch**.

You can now assign an Elastic IP address to the Amazon EC2 instance that you just created. You need to use this IP address to connect to the Amazon EC2 instance.

**To assign an elastic IP address (Console)**

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation list, choose **Elastic IPs**.
3. Choose **Allocate Elastic IP address**.

4. In the **Allocate Elastic IP address** dialog box, accept the default **Network Border Group** and choose **Allocate** .
5. Choose the Elastic IP address that you just allocated from the list and choose **Associate Address**.
6. In the **Associate Address** dialog box, in the **Instance** box, choose the ID of the Amazon EC2 instance that you launched.

   In the **Private IP address** box, select the box to obtain the private IP address and then choose **Associate**.

   You can now use SSH to connect to the Amazon EC2 instance using the Elastic IP address that you created.

**To connect to your Amazon EC2 instance**

- Open a command window. At the command prompt, issue the following command, replacing *mykeypair.pem* with the name of your key pair file and *54.207.55.251* with your Elastic IP address.

```
ssh -i mykeypair.pem ec2-user@54.207.55.251
```

> **Important**
>
> Do not log out of your Amazon EC2 instance yet.

You are now ready to interact with your ElastiCache cluster. Before you can do that, if you haven't already done so, you need to install the *telnet* utility.

**To install *telnet* and interact with your cluster (AWS CLI)**

- Open a command window. At the command prompt, issue the following command. At the confirmation prompt, type *y*.

```
sudo yum install telnet
Loaded plugins: priorities, security, update-motd, upgrade-helper
Setting up Install Process
```

```
Resolving Dependencies
--> Running transaction check

...(output omitted)...

Total download size: 63 k
Installed size: 109 k
Is this ok [y/N]: y
Downloading Packages:
telnet-0.17-47.7.amzn1.x86_64.rpm                          |  63 kB     00:00

...(output omitted)...

Complete!
```

You can now connect to a VPC with either Memcached or Redis.

### Connecting to a VPC with Memcached

1. Go to the ElastiCache console at https://console.aws.amazon.com/elasticache/ and obtain the endpoint for one of the nodes in your cluster. For more information, see Finding connection endpoints.
2. Use *telnet* to connect to your cache node endpoint over port 11211. Replace the hostname shown below with the hostname of your cache node.

   ```
   telnet my-cache-cluster.7wufxa.0001.use1.cache.amazonaws.com 11211
   ```

   You are now connected to the cache engine and can issue commands. In this example, you add a data item to the cache and then get it immediately afterward. Finally, you'll disconnect from the cache node.

   To store a key and a value, type the following two lines:

   ```
   add mykey 0 3600 28
   This is the value for mykey
   ```

   The cache engine responds with the following:

```
OK
```

To retrieve the value for `mykey`, type the following:

```
get mykey
```

The cache engine responds with the following:

```
VALUE mykey 0 28
This is the value for my key
END
```

To disconnect from the cache engine, type the following:

```
quit
```

### Connecting to a VPC with Redis

1. Go to the ElastiCache console at https://console.aws.amazon.com/elasticache/ and obtain the endpoint for one of the nodes in your cluster. For more information, see Finding connection endpoints for Redis.
2. Use *telnet* to connect to your cache node endpoint over port 6379. Replace the hostname shown below with the hostname of your cache node.

   ```
   telnet my-cache-cluster.7wufxa.0001.use1.cache.amazonaws.com 6379
   ```

   You are now connected to the cache engine and can issue commands. In this example, you add a data item to the cache and then get it immediately afterward. Finally, you'll disconnect from the cache node.

   To store a key and a value, type the following two lines:

   ```
   set mykey myvalue
   ```

   The cache engine responds with the following:

```
OK
```

To retrieve the value for `mykey`, type the following:

```
get mykey
```

To disconnect from the cache engine, type the following:

```
quit
```

3. Go to the ElastiCache console at https://console.aws.amazon.com/elasticache/ and obtain the endpoint for one of the nodes in your cluster. For more information, Finding connection endpoints for Redis OSS.
4. Use *telnet* to connect to your cache node endpoint over port 6379. Replace the hostname shown below with the hostname of your cache node.

```
telnet my-cache-cluster.7wufxa.0001.use1.cache.amazonaws.com 6379
```

You are now connected to the cache engine and can issue commands. In this example, you add a data item to the cache and then get it immediately afterward. Finally, you'll disconnect from the cache node.

To store a key and a value, type the following:

```
set mykey myvalue
```

The cache engine responds with the following:

```
OK
```

To retrieve the value for `mykey`, type the following:

```
get mykey
```

The cache engine responds with the following:

```
get mykey
myvalue
```

To disconnect from the cache engine, type the following:

```
quit
```

> **Important**
>
> To avoid incurring additional charges on your AWS account, be sure to delete any AWS resources you no longer want after trying these examples.

## ElastiCache API and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon ElastiCache API endpoints by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink. AWS PrivateLink allows you to privately access Amazon ElastiCache API operations without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection.

Instances in your VPC don't need public IP addresses to communicate with Amazon ElastiCache API endpoints. Your instances also don't need public IP addresses to use any of the available ElastiCache API operations. Traffic between your VPC and Amazon ElastiCache doesn't leave the Amazon network. Each interface endpoint is represented by one or more elastic network interfaces in your subnets. For more information on elastic network interfaces, see Elastic network interfaces in the *Amazon EC2 User Guide*.

- For more information about VPC endpoints, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.
- For more information about ElastiCache API operations, see ElastiCache API operations.

After you create an interface VPC endpoint, if you enable private DNS hostnames for the endpoint, the default ElastiCache endpoint (https://elasticache.*Region*.amazonaws.com) and dualstack endpoint (https://elasticache.*Region*.api.aws) resolve to your VPC endpoint. If you do not enable

private DNS hostnames, Amazon VPC provides a DNS endpoint name that you can use in the following format:

```
VPC_Endpoint_ID.elasticache.Region.vpce.amazonaws.com
```

For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*. ElastiCache supports making calls to all of its API Actions inside your VPC.

> **Note**
>
> Private DNS hostnames can be enabled for only one VPC endpoint in the VPC. If you want to create an additional VPC endpoint then private DNS hostname should be disabled for that endpoint.

### Available Privatelink Regions

| Code | Location | Region |
|---|---|---|
| CPT | Africa (Cape Town) | AF-SOUTH-1 |
| HKG | Asia Pacific (Hong Kong) | AP-EAST-1 |
| TPE | Asia Pacific (Taipei) | AP-EAST-2 |
| NRT | Asia Pacific (Tokyo) | AP-NORTHEAST-1 |
| ICN | Asia Pacific (Seoul) | AP-NORTHEAST-2 |
| KIX | Asia Pacific (Osaka) | AP-NORTHEAST-3 |
| BOM | Asia Pacific (Mumbai) | AP-SOUTH-1 |
| HYD | Asia Pacific (Hyderabad) | AP-SOUTH-2 |
| SIN | Asia Pacific (Singapore) | AP-SOUTHEAST-1 |
| SYD | Asia Pacific (Sydney) | AP-SOUTHEAST-2 |
| CGK | Asia Pacific (Jakarta) | AP-SOUTHEAST-3 |

| Code | Location | Region |
|---|---|---|
| MEL | Asia Pacific (Melbourne) | AP-SOUTHEAST-4 |
| KUL | Asia Pacific (Malaysia) | AP-SOUTHEAST-5 |
| BKK | Asia Pacific (Thailand) | AP-SOUTHEAST-7 |
| YUL | Canada (Central) | CA-CENTRAL-1 |
| YYC | Canada West (Calgary) | CA-WEST-1 |
| BJS | China (Beijing) | CN-NORTH-1 |
| ZHY | China (Ningxia) | CN-NORTHWEST-1 |
| FRA | Europe (Frankfurt) | EU-CENTRAL-1 |
| ZRH | Europe (Zurich) | EU-CENTRAL-2 |
| ARN | Europe (Stockholm) | EU-NORTH-1 |
| MXP | Europe (Milan) | EU-SOUTH-1 |
| ZAZ | Europe (Spain) | EU-SOUTH-2 |
| DUB | Europe (Ireland) | EU-WEST-1 |
| LHR | Europe (London) | EU-WEST-2 |
| CDG | Europe (Paris) | EU-WEST-3 |
| TLV | Tel Aviv (Israel) | IL-CENTRAL-1 |
| DXB | Middle East (UAE) | ME-CENTRAL-1 |
| BAH | Middle East (Bahrain) | ME-SOUTH-1 |
| QRO | Mexico (Central) | MX-CENTRAL-1 |
| GRU | South America (Sao Paulo) | SA-EAST-1 |

| Code | Location | Region |
|---|---|---|
| IAD | US East (N. Virginia) | US-EAST-1 |
| CMH | US East (Ohio) | US-EAST-2 |
| OSU | AWS GovCloud (US-East) | US-GOV-EAST-1 |
| SFO | US West (N. California) | US-WEST-1 |
| PDX | US West (Oregon) | US-WEST-2 |
| PDT | AWS GovCloud (US-West) | US-WEST-1 |

## Considerations for VPC endpoints

Before you set up an interface VPC endpoint for Amazon ElastiCache API endpoints, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*. All ElastiCache API operations relevant to managing Amazon ElastiCache resources are available from your VPC using AWS PrivateLink.

VPC endpoint policies are supported for ElastiCache API endpoints. By default, full access to ElastiCache API operations is allowed through the endpoint. For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

## Creating an interface VPC endpoint for the ElastiCache API

You can create a VPC endpoint for the Amazon ElastiCache API using either the Amazon VPC console or the AWS CLI. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

After you create an interface VPC endpoint, you can enable private DNS hostnames for the endpoint. When you do, the default Amazon ElastiCache endpoint (https://elasticache.*Region*.amazonaws.com) and dualstack endpoint (https://elasticache.*Region*.api.aws) resolve to your VPC endpoint. For the China (Beijing) and China (Ningxia) AWS Regions, you can make API requests with the VPC endpoint by using `elasticache.cn-north-1.amazonaws.com.cn` and `elasticache.cn-north-1.api.amazonwebservices.com.cn` for Beijing, and `elasticache.cn-northwest-1.amazonaws.com.cn` and `elasticache.cn-`

`northwest-1.api.amazonwebservices.com.cn` for Ningxia. For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

## Creating a VPC endpoint policy for the Amazon ElastiCache API

You can attach an endpoint policy to your VPC endpoint that controls access to the ElastiCache API. The policy specifies the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

### Example VPC endpoint policy for ElastiCache API actions with Valkey or Redis OSS

The following is an example of an endpoint policy for the ElastiCache API. When attached to an endpoint, this policy grants access to the listed ElastiCache API actions for all principals on all resources.

```
{
 "Statement": [{
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
   "elasticache:CreateCacheCluster",
   "elasticache:ModifyCacheCluster",
   "elasticache:CreateSnapshot"
  ],
  "Resource": "*"
 }]
}
```

### Example VPC endpoint policy for ElastiCache for Memcached API actions

The following is an example of an endpoint policy for the ElastiCache API. When attached to an endpoint, this policy grants access to the listed ElastiCache API actions for all principals on all resources.

```
{
 "Statement": [{
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
   "elasticache:CreateCacheCluster",
   "elasticache:ModifyCacheCluster"
  ],
  "Resource": "*"
 }]
}
```

**Example VPC endpoint policy that denies all access from a specified AWS account**

The following VPC endpoint policy denies AWS account *123456789012* all access to resources using the endpoint. The policy allows all actions from other accounts.

```
{
 "Statement": [{
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*",
   "Principal": "*"
  },
  {
   "Action": "*",
   "Effect": "Deny",
   "Resource": "*",
   "Principal": {
    "AWS": [
     "123456789012"
    ]
   }
  }
 ]
}
```

## Subnets and subnet groups

A *subnet group* is a collection of subnets (typically private) that you can designate for your node-based clusters running in an Amazon Virtual Private Cloud (VPC) environment.

If you create a node-based cluster in an Amazon VPC, you must use a subnet group. ElastiCache uses that subnet group to choose a subnet and IP addresses within that subnet to associate with your nodes.

ElastiCache provides a default IPv4 subnet group or you can choose to create a new one. For IPv6, you need to create a subnet group with an IPv6 CIDR block. If you choose **dual stack**, you then must select a Discovery IP type, either IPv6 or IPv4.

ElastiCache Serverless does not use a subnet group resource, and instead takes a list of subnets directly during creation.

This section covers how to create and leverage subnets and subnet groups to manage access to your ElastiCache resources.

For more information about subnet group usage in an Amazon VPC environment, see Accessing your ElastiCache cluster or replication group.

**Topics**


- Creating a subnet group
- Assigning a subnet group to a cache
- Modifying a subnet group
- IP address capacity for scaling operations
- Deleting a subnet group

## Creating a subnet group

A *cache subnet group* is a collection of subnets that you may want to designate for your caches in a VPC. When launching a cache in a VPC, you need to select a cache subnet group. Then ElastiCache uses that cache subnet group to assign IP addresses within that subnet to each cache node in the cache.

When you create a new subnet group, note the number of available IP addresses. If the subnet has very few free IP addresses, you might be constrained as to how many more nodes you can add to a cluster. To resolve this issue, you can assign one or more subnets to a subnet group so that you have a sufficient number of IP addresses in your cluster's Availability Zone. After that, you can add more nodes to your cluster.

If you choose IPV4 as your network type, a default subnet group will be available or you can choose to create a new one. ElastiCache uses that subnet group to choose a subnet and IP addresses within that subnet to associate with your nodes. If you choose dual-stack or IPV6, you will be directed to create dual-stack or IPV6 subnets. For more information on network types, see Choosing a network type in ElastiCache. For more information, see Create a subnet in your VPC.

The following procedures show you how to create a subnet group called `mysubnetgroup` (console), the AWS CLI, and the ElastiCache API.

> **Subnet group name casing**
>
> Amazon ElastiCache converts subnet group names to lowercase at creation time. For example, if you specify `MySubnetGroup`, the resulting name is `mysubnetgroup`. To avoid update failures, use lowercase names in your infrastructure-as-code templates.

### Creating a subnet group (Console)

The following procedure shows how to create a subnet group (console).

**To create a subnet group (Console)**

1. Sign in to the AWS Management Console, and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation list, choose **Subnet groups**.
3. Choose **Create subnet group**.

4. In the **Create subnet group** wizard, do the following. When all the settings are as you want them, choose **Create**.

   a. In the **Name** box, type a name for your subnet group.

   b. In the **Description** box, type a description for your subnet group.

   c. In the **VPC ID** box, choose your Amazon VPC.

   d. All subnets are chosen by default. In the **Selected subnets** panel, click **Manage** and select the Availability Zones or Local Zones and IDs of your private subnets, and then choose **Choose**.

5. In the confirmation message that appears, choose **Close**.

Your new subnet group appears in the **Subnet Groups** list of the ElastiCache console. At the bottom of the window you can choose the subnet group to see details, such as all of the subnets associated with this group.

**Creating a subnet group (AWS CLI)**

At a command prompt, use the command `create-cache-subnet-group` to create a subnet group.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-subnet-group \
    --cache-subnet-group-name mysubnetgroup \
    --cache-subnet-group-description "Testing" \
    --subnet-ids subnet-53df9c3a
```

For Windows:

```
aws elasticache create-cache-subnet-group ^
    --cache-subnet-group-name mysubnetgroup ^
    --cache-subnet-group-description "Testing" ^
    --subnet-ids subnet-53df9c3a
```

This command should produce output similar to the following:

```
{
    "CacheSubnetGroup": {
        "VpcId": "vpc-37c3cd17",
```

```
        "CacheSubnetGroupDescription": "Testing",
        "Subnets": [
            {
                "SubnetIdentifier": "subnet-53df9c3a",
                "SubnetAvailabilityZone": {
                    "Name": "us-west-2a"
                }
            }
        ],
        "CacheSubnetGroupName": "mysubnetgroup"
    }
}
```

For more information, see the AWS CLI topic create-cache-subnet-group.

## Assigning a subnet group to a cache

After you have created a subnet group, you can launch a cache in an Amazon VPC. For more information, see the following.

- **Memcached cluster** – To launch a Memcached cluster, see Creating a Memcached cluster (console). In step 7.a (**Advanced Memcached Settings**), choose a VPC subnet group.
- **Standalone Valkey or Redis OSS cluster** – To launch a single-node Valkey or Redis OSS cluster, see Creating a Valkey (cluster mode disabled) cluster (Console). In step 7.a (**Advanced Redis OSS Settings**), choose a VPC subnet group.
- **Valkey or Redis OSS (cluster mode disabled) replication group** – To launch a Valkey or Redis OSS (cluster mode disabled) replication group in a VPC, see Creating a Valkey or Redis OSS (Cluster Mode Disabled) replication group from scratch. In step 7.b (**Advanced Redis OSS Settings**), choose a VPC subnet group.
- **Valkey or Redis OSS (cluster mode enabled) replication group** – Creating a Valkey or Redis OSS (Cluster Mode Enabled) cluster (Console). In step 6.i (**Advanced Redis OSS Settings**), choose a VPC subnet group.

## Modifying a subnet group

You can modify a subnet group's description, or modify the list of subnet IDs associated with the subnet group. You cannot delete a subnet ID from a subnet group if a cache is currently using that subnet.

The following procedures show you how to modify a subnet group.

### Modifying subnet groups (Console)

#### To modify a subnet group

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Subnet groups**.
3. In the list of subnet groups, select the radio button of the one you want to modify and choose **Modify**.
4. In the **Selected subnets** panel, choose **Manage**.
5. Make any changes to the selected subnets and click **Choose**.
6. Click **Save changes** to save your changes.

### Modifying subnet groups (AWS CLI)

At a command prompt, use the command `modify-cache-subnet-group` to modify a subnet group.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-subnet-group \
    --cache-subnet-group-name mysubnetgroup \
    --cache-subnet-group-description "New description" \
    --subnet-ids "subnet-42df9c3a" "subnet-48fc21a9"
```

For Windows:

```
aws elasticache modify-cache-subnet-group ^
    --cache-subnet-group-name mysubnetgroup ^
    --cache-subnet-group-description "New description" ^
```

```
    --subnet-ids "subnet-42df9c3a" "subnet-48fc21a9"
```

This command should produce output similar to the following:

```
{
    "CacheSubnetGroup": {
        "VpcId": "vpc-73cd3c17",
        "CacheSubnetGroupDescription": "New description",
        "Subnets": [
            {
                "SubnetIdentifier": "subnet-42dcf93a",
                "SubnetAvailabilityZone": {
                    "Name": "us-west-2a"
                }
            },
            {
                "SubnetIdentifier": "subnet-48fc12a9",
                "SubnetAvailabilityZone": {
                    "Name": "us-west-2a"
                }
            }
        ],
        "CacheSubnetGroupName": "mysubnetgroup"
    }
}
```

For more information, see the AWS CLI topic modify-cache-subnet-group.

## IP address capacity for scaling operations

When you scale out a cluster by adding shards or replicas, Amazon ElastiCache provisions new nodes. Each node requires an available IP address in a subnet within the target Availability Zone. If the subnets in an Availability Zone don't have enough free IP addresses, the scaling operation fails.

To avoid this issue, review the available IP address capacity of your subnet group's subnets before you scale. To check the number of available IP addresses for each subnet, use the Amazon Elastic Compute Cloud (Amazon EC2) `DescribeSubnets` API. You can also view this information in the Amazon VPC console.

If a subnet is approaching IP address exhaustion, take one of the following actions before you scale:

- Add a subnet with sufficient available IP addresses to the subnet group in the same Availability Zone.
- Remove unused resources, such as unattached network interfaces, to free IP addresses in the existing subnet.
- Replace the subnet with one that uses a larger CIDR block.

## Deleting a subnet group

If you decide that you no longer need your subnet group, you can delete it. You cannot delete a subnet group if it is currently in use by a cache.

The following procedures show you how to delete a subnet group.

### Deleting a subnet group (Console)

#### To delete a subnet group

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation pane, choose **Subnet groups**.
3. In the list of subnet groups, choose the one you want to delete and then choose **Delete**.
4. When you are asked to confirm this operation, type the name of the subnet group in the text input field and choose **Delete**.

### Deleting a subnet group (AWS CLI)

Using the AWS CLI, call the command **delete-cache-subnet-group** with the following parameter:

- `--cache-subnet-group-name` *`mysubnetgroup`*

For Linux, macOS, or Unix:

```
aws elasticache delete-cache-subnet-group \
    --cache-subnet-group-name mysubnetgroup
```

For Windows:

```
aws elasticache delete-cache-subnet-group ^
    --cache-subnet-group-name mysubnetgroup
```

This command produces no output.

For more information, see the AWS CLI topic delete-cache-subnet-group.

# Identity and Access Management for Amazon ElastiCache

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use ElastiCache resources. IAM is an AWS service that you can use with no additional charge.

**Topics**


- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon ElastiCache works with IAM
- Identity-based policy examples for Amazon ElastiCache
- Troubleshooting Amazon ElastiCache identity and access
- Access control
- Overview of managing access permissions to your ElastiCache resources


## Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon ElastiCache identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon ElastiCache works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for Amazon ElastiCache)

## Authenticating with identities

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*.

### AWS account root user

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*.

### Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

### IAM users and groups

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*.

### IAM roles

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*.

## Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

### Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

### Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-

based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

### Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

### Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

## How Amazon ElastiCache works with IAM

Before you use IAM to manage access to ElastiCache, learn what IAM features are available to use with ElastiCache.

**IAM features you can use with Amazon ElastiCache**

| IAM feature | ElastiCache support |
|---|---|
| Identity-based policies | Yes |
| Resource-based policies | No |
| Policy actions | Yes |
| Policy resources | Yes |
| Policy condition keys | Yes |
| ACLs | Yes |
| ABAC (tags in policies) | Yes |
| Temporary credentials | Yes |
| Principal permissions | Yes |
| Service roles | Yes |
| Service-linked roles | Yes |

To get a high-level view of how ElastiCache and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

## Identity-based policies for ElastiCache

**Supports identity-based policies:** Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

### Identity-based policy examples for ElastiCache

To view examples of ElastiCache identity-based policies, see Identity-based policy examples for Amazon ElastiCache.

## Resource-based policies within ElastiCache

**Supports resource-based policies:** No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

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. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

## Policy actions for ElastiCache

**Supports policy actions:** Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of ElastiCache actions, see Actions Defined by Amazon ElastiCache in the *Service Authorization Reference*.

Policy actions in ElastiCache use the following prefix before the action:

```
elasticache
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "elasticache:action1",
      "elasticache:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "elasticache:Describe*"
```

To view examples of ElastiCache identity-based policies, see Identity-based policy examples for Amazon ElastiCache.

## Policy resources for ElastiCache

**Supports policy resources:** Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of ElastiCache resource types and their ARNs, see Resources Defined by Amazon ElastiCache in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon ElastiCache.

To view examples of ElastiCache identity-based policies, see Identity-based policy examples for Amazon ElastiCache.

## Policy condition keys for ElastiCache

**Supports service-specific policy condition keys:** Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of ElastiCache condition keys, see Condition Keys for Amazon ElastiCache in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon ElastiCache.

To view examples of ElastiCache identity-based policies, see Identity-based policy examples for Amazon ElastiCache.

## Access control lists (ACLs) in ElastiCache

**Supports ACLs:** Yes

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

## Attribute-based access control (ABAC) with ElastiCache

**Supports ABAC (tags in policies):** Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

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 ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

## Using Temporary credentials with ElastiCache

**Supports temporary credentials:** Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

## Cross-service principal permissions for ElastiCache

**Supports forward access sessions (FAS):** Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

## Service roles for ElastiCache

**Supports service roles:** Yes

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*.

> **Warning**
>
> Changing the permissions for a service role might break ElastiCache functionality. Edit service roles only when ElastiCache provides guidance to do so.

## Service-linked roles for ElastiCache

**Supports service-linked roles:** Yes

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.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

## Identity-based policy examples for Amazon ElastiCache

By default, users and roles don't have permission to create or modify ElastiCache resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by ElastiCache, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon ElastiCache in the *Service Authorization Reference*.

**Topics**


- Policy best practices
- Using the ElastiCache console
- Allow users to view their own permissions


### Policy best practices

Identity-based policies determine whether someone can create, access, or delete ElastiCache resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **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. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **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*. For more

information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

## Using the ElastiCache console

To access the Amazon ElastiCache console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the ElastiCache resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the ElastiCache console, also attach the ElastiCache `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

## Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

# Troubleshooting Amazon ElastiCache identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with ElastiCache and IAM.

**Topics**


- I am not authorized to perform an action in ElastiCache
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my ElastiCache resources


## I am not authorized to perform an action in ElastiCache

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `elasticache:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 elasticache:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `elasticache:`*`GetWidget`* action.

## I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to ElastiCache.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in ElastiCache. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

### I want to allow people outside of my AWS account to access my ElastiCache resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether ElastiCache supports these features, see How Amazon ElastiCache works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

## Access control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access ElastiCache resources. For example, you must have permissions to create an ElastiCache cluster.

The following sections describe how to manage permissions for ElastiCache. We recommend that you read the overview first.

- Overview of managing access permissions to your ElastiCache resources
- Using identity-based policies (IAM policies) for Amazon ElastiCache

# Overview of managing access permissions to your ElastiCache resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). In addition, Amazon ElastiCache also supports attaching permissions policies to resources.

> **Note**
>
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

**Topics**


- Amazon ElastiCache resources and operations
- Understanding resource ownership
- Managing access to resources
- AWS managed policies for Amazon ElastiCache
- Using identity-based policies (IAM policies) for Amazon ElastiCache

- Resource-level permissions
- Using condition keys
- Using Service-Linked Roles for Amazon ElastiCache
- ElastiCache API permissions: Actions, resources, and conditions reference

## Amazon ElastiCache resources and operations

To see a list of ElastiCache resource types and their ARNs, see Resources Defined by Amazon ElastiCache in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon ElastiCache.

## Understanding resource ownership

A *resource owner* is the AWS account that created the resource. That is, the resource owner is the AWS account of the principal entity that authenticates the request that creates the resource. A *principal entity* can be the root account, an IAM user, or an IAM role). The following examples illustrate how this works:

- Suppose that you use the root account credentials of your AWS account to create a cache cluster. In this case, your AWS account is the owner of the resource. In ElastiCache, the resource is the cluster.
- Suppose that you create an IAM user in your AWS account and grant permissions to create a cluster to that user. In this case, the user can create a cluster. However, your AWS account, to which the user belongs, owns the cluster resource.
- Suppose that you create an IAM role in your AWS account with permissions to create a cluster. In this case, anyone who can assume the role can create a cluster. Your AWS account, to which the role belongs, owns the cache cluster resource.

## Managing access to resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

> **Note**
>
> This section discusses using IAM in the context of Amazon ElastiCache. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is

IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM policies). Policies attached to a resource are referred to as *resource-based* policies.

**Topics**

- Identity-based policies (IAM policies)
- Specifying policy elements: Actions, effects, resources, and principals
- Specifying conditions in a policy

### Identity-based policies (IAM policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – An account administrator can use a permissions policy that is associated with a particular user to grant permissions. In this case, the permissions are for that user to create an ElastiCache resource, such as a cluster, parameter group, or security group.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. In some cases, you might want to grant an AWS service permissions to assume the role. To support this approach, the principal in the trust policy can also be an AWS service principal.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following is an example policy that allows a user to perform the `DescribeCacheClusters` action for your AWS account. ElastiCache also supports identifying specific resources using the resource ARNs for API actions. (This approach is also referred to as resource-level permissions).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeCacheClusters",
            "Effect": "Allow",
            "Action": [
                "elasticache:DescribeCacheClusters"
            ],
            "Resource": "arn:aws:iam::*:role/*"
        }
    ]
}
```

For more information about using identity-based policies with ElastiCache, see Using identity-based policies (IAM policies) for Amazon ElastiCache. For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles in the *IAM User Guide*.

**Specifying policy elements: Actions, effects, resources, and principals**

For each Amazon ElastiCache resource (see Amazon ElastiCache resources and operations), the service defines a set of API operations (see Actions). To grant permissions for these API operations, ElastiCache defines a set of actions that you can specify in a policy. For example, for the ElastiCache cluster resource, the following actions are defined: `CreateCacheCluster`, `DeleteCacheCluster`, and `DescribeCacheCluster`. Performing an API operation can require permissions for more than one action.

The following are the most basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies. For more information, see Amazon ElastiCache resources and operations.

- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, depending on the specified `Effect`, the `elasticache:CreateCacheCluster` permission allows or denies the user permissions to perform the Amazon ElastiCache `CreateCacheCluster` operation.
- **Effect** – You specify the effect when the user requests the specific action—this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. For example, you might do this to make sure that a user can't access a resource, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only).

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the Amazon ElastiCache API actions, see ElastiCache API permissions: Actions, resources, and conditions reference.

**Specifying conditions in a policy**

When you grant permissions, you can use the IAM policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. To use ElastiCache-specific condition keys, see Using condition keys. There are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

## AWS managed policies for Amazon ElastiCache

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*.

### AWS managed policy: ElastiCacheServiceRolePolicy

You can't attach ElastiCacheServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows ElastiCache to perform actions on your behalf.

This policy allows ElastiCache to manage AWS resources on your behalf as necessary for managing your cache:

- `ec2` – Manage EC2 networking resources to attach to cache nodes, including VPC endpoints (for serverless caches), Elastic Network Interfaces (ENIs) (for node-based clusters), and security groups.
- `cloudwatch` – Emit metric data from the service into CloudWatch.
- `outposts` – Allow creation of cache nodes on AWS Outposts.

You can find the ElastiCacheServiceRolePolicy policy on the IAM console and ElastiCacheServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

**AWS managed policy: AmazonElastiCacheFullAccess**

You can attach the `AmazonElastiCacheFullAccess` policy to your IAM identities.

This policy allows principals full access to ElastiCache using the AWS Management Console:

- `elasticache` — Access all APIs.
- `iam` — Create service-linked role necessary for service operation.
- `ec2` — Describe dependent EC2 resources necessary for cache creation (VPC, subnet, security group) and allow creation of VPC endpoints (for serverless caches).
- `kms` — Allow usage of customer-managed CMKs for encryption-at-rest.
- `cloudwatch` — Allow access to metrics to display ElastiCache metrics in the console.
- `application-autoscaling` — Allow access to describe autoscaling policies for caches.
- `logs` — Used to populate log streams for log delivery functionality in the console.
- `firehose` — Used to populate delivery streams for log delivery functionality in the console.
- `s3` — Used to populate S3 buckets for snapshot restore functionality in the console.
- `outposts` — Used to populate AWS Outposts for cache creation in the console.
- `sns` — Used to populate SNS topics for notification functionality in the console.

You can find the AmazonElastiCacheFullAccess policy on the IAM console and AmazonElastiCacheFullAccess in the *AWS Managed Policy Reference Guide*.

**AWS managed policy: AmazonElastiCacheReadOnlyAccess**

You can attach the `AmazonElastiCacheReadOnlyAccess` policy to your IAM identities.

This policy allows principals read-only access to ElastiCache using the AWS Management Console:

- `elasticache` — Access read-only `Describe` APIs.

You can find the AmazonElastiCacheReadOnlyAccess policy on the IAM console and AmazonElastiCacheReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

**ElastiCache updates to AWS managed policies**

View details about updates to AWS managed policies for ElastiCache since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the ElastiCache Document history page.

| Change | Description | Date |
| --- | --- | --- |
| AmazonElastiCacheFullAccess – Update to an existing policy | ElastiCache added new permissions to allow vertical scaling for MemCached, for the action `elasticache:ModifyCacheCluster` . | March 27, 2025 |
| AmazonElastiCacheFullAccess – Update to an existing policy | ElastiCache added new permissions to allow management of serverless caches, and to enable usage of all service features via the console. | November 27, 2023 |
| ElastiCacheServiceRolePolicy – Update to an existing policy | ElastiCache added new permissions to allow management of VPC endpoints for serverless cache resources. | November 27, 2023 |
| ElastiCache started tracking changes | ElastiCache started tracking changes for its AWS managed policies. | February 07, 2020 |

## Using identity-based policies (IAM policies) for Amazon ElastiCache

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

> **Important**
>
> We recommend that you first read the topics that explain the basic concepts and options to manage access to Amazon ElastiCache resources. For more information, see Overview of managing access permissions to your ElastiCache resources.

The sections in this topic cover the following:

- AWS managed policies for Amazon ElastiCache
- Customer-managed policy examples

The following shows an example of a permissions policy when using Redis OSS.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowClusterPermissions",
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateServerlessCache",
                "elasticache:CreateCacheCluster",
                "elasticache:DescribeServerlessCaches",
                "elasticache:DescribeReplicationGroups",
                "elasticache:DescribeCacheClusters",
                "elasticache:ModifyServerlessCache",
                "elasticache:ModifyReplicationGroup",
                "elasticache:ModifyCacheCluster"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowUserToPassRole",
            "Effect": "Allow",
            "Action": [ "iam:PassRole" ],
            "Resource": "arn:aws:iam::123456789012:role/EC2-roles-for-cluster"
        }
    ]
}
```

The following shows an example of a permissions policy when using Memcached.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "AllowClusterPermissions",
        "Effect": "Allow",
        "Action": [
            "elasticache:CreateServerlessCache",
            "elasticache:CreateCacheCluster",
            "elasticache:DescribeServerlessCaches",
            "elasticache:DescribeCacheClusters",
            "elasticache:ModifyServerlessCache",
            "elasticache:ModifyCacheCluster"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AllowUserToPassRole",
        "Effect": "Allow",
        "Action": [ "iam:PassRole" ],
        "Resource": "arn:aws:iam::123456789012:role/EC2-roles-for-cluster"
    }
    ]
}
```

The policy has two statements:

- The first statement grants permissions for the Amazon ElastiCache actions (`elasticache:Create*`, `elasticache:Describe*`, `elasticache:Modify*`)
- The second statement grants permissions for the IAM action (`iam:PassRole`) on the IAM role name specified at the end of the `Resource` value.

The policy doesn't specify the `Principal` element because in an identity-based policy you don't specify the principal who gets the permission. When you attach policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon ElastiCache API actions and the resources that they apply to, see ElastiCache API permissions: Actions, resources, and conditions reference.

### Customer-managed policy examples

If you are not using a default policy and choose to use a custom-managed policy, ensure one of two things. Either you should have permissions to call `iam:createServiceLinkedRole` (for more information, see Example 4: Allow a user to call IAM CreateServiceLinkedRole API). Or you should have created an ElastiCache service-linked role.

When combined with the minimum permissions needed to use the Amazon ElastiCache console, the example policies in this section grant additional permissions. The examples are also relevant to the AWS SDKs and the AWS CLI.

For instructions on setting up IAM users and groups, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.

> **Important**
>
> Always test your IAM policies thoroughly before using them in production. Some ElastiCache actions that appear simple can require other actions to support them when you are using the ElastiCache console. For example, `elasticache:CreateCacheCluster` grants permissions to create ElastiCache clusters. However, to perform this operation, the ElastiCache console uses a number of `Describe` and `List` actions to populate console lists.

### Examples

- Example 1: Allow a user read-only access to ElastiCache resources
- Example 2: Allow a user to perform common ElastiCache system administrator tasks
- Example 3: Allow a user to access all ElastiCache API actions
- Example 4: Allow a user to call IAM CreateServiceLinkedRole API
- Example 5: Allow a user to connect to serverless cache using IAM authentication

### Example 1: Allow a user read-only access to ElastiCache resources

The following policy grants permissions ElastiCache actions that allow a user to list resources. Typically, you attach this type of permissions policy to a managers group.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[{
      "Sid": "ECReadOnly",
      "Effect":"Allow",
      "Action": [
          "elasticache:Describe*",
          "elasticache:List*"],
      "Resource":"*"
      }
   ]
}
```

**Example 2: Allow a user to perform common ElastiCache system administrator tasks**

Common system administrator tasks include modifying resources. A system administrator may also want to get information about the ElastiCache events. The following policy grants a user permissions to perform ElastiCache actions for these common system administrator tasks. Typically, you attach this type of permissions policy to the system administrators group.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[{
      "Sid": "ECAllowMutations",
      "Effect":"Allow",
      "Action":[
          "elasticache:Modify*",
          "elasticache:Describe*",
          "elasticache:ResetCacheParameterGroup"
      ],
      "Resource":"*"
      }
   ]
}
```

### Example 3: Allow a user to access all ElastiCache API actions

The following policy allows a user to access all ElastiCache actions. We recommend that you grant this type of permissions policy only to an administrator user.

JSON

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

### Example 4: Allow a user to call IAM CreateServiceLinkedRole API

The following policy allows user to call the IAM `CreateServiceLinkedRole` API. We recommend that you grant this type of permissions policy to the user who invokes mutative ElastiCache operations.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"CreateSLRAllows",
      "Effect":"Allow",
      "Action":[
        "iam:CreateServiceLinkedRole"
      ],
      "Resource":"*",
      "Condition":{
        "StringLike":{
```

```
        "iam:AWSServiceName":"elasticache.amazonaws.com"
        }
      }
    }
  ]
}
```

### Example 5: Allow a user to connect to serverless cache using IAM authentication

The following policy allows any user to connect to any serverless cache using IAM authentication between 2023-04-01 and 2023-06-30.

JSON

```
{
  "Version":"2012-10-17",
  "Statement" :
  [
    {
      "Effect" : "Allow",
      "Action" : ["elasticache:Connect"],
      "Resource" : [
        "arn:aws:elasticache:us-east-1:123456789012:serverlesscache:*"
      ],
      "Condition": {
        "DateGreaterThan": {"aws:CurrentTime": "2023-04-01T00:00:00Z"},
        "DateLessThan": {"aws:CurrentTime": "2023-06-30T23:59:59Z"}
      }
    },
    {
      "Effect" : "Allow",
      "Action" : ["elasticache:Connect"],
      "Resource" : [
        "arn:aws:elasticache:us-east-1:123456789012:user:*"
      ]
    }
  ]
}
```

## Resource-level permissions

You can restrict the scope of permissions by specifying resources in an IAM policy. Many ElastiCache API actions support a resource type that varies depending on the behavior of the action. Every IAM policy statement grants permission to an action that's performed on a resource. When the action doesn't act on a named resource, or when you grant permission to perform the action on all resources, the value of the resource in the policy is a wildcard (*). For many API actions, you can restrict the resources that a user can modify by specifying the Amazon Resource Name (ARN) of a resource, or an ARN pattern that matches multiple resources. To restrict permissions by resource, specify the resource by ARN.

> **Note**
>
> For resource-level permissions to be effective, the resource name in the ARN string should be lowercase.

To see a list of ElastiCache resource types and their ARNs, see Resources Defined by Amazon ElastiCache in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon ElastiCache.

### Examples

- Example 1: Allow a user full access to specific ElastiCache resource types
- Example 2: Deny a user access to a serverless cache.

### Example 1: Allow a user full access to specific ElastiCache resource types

The following policy explicitly allows all resources of type serverless cache.

```
{
        "Sid": "Example1",
        "Effect": "Allow",
        "Action": "elasticache:*",
        "Resource": [
             "arn:aws:elasticache:us-east-1:account-id:serverlesscache:*"
        ]
}
```

**Example 2: Deny a user access to a serverless cache.**

The following example explicitly denies access to a particular serverless cache.

```
{
        "Sid": "Example2",
        "Effect": "Deny",
        "Action": "elasticache:*",
        "Resource": [
            "arn:aws:elasticache:us-east-1:account-id:serverlesscache:name"
        ]
}
```

## Using condition keys

You can specify conditions that determine how an IAM policy takes effect. In ElastiCache, 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. For more information, see IAM JSON policy elements: Condition.

To see a list of ElastiCache condition keys, see Condition Keys for Amazon ElastiCache in the *Service Authorization Reference*.

For a list of global condition keys, see AWS global condition context keys.

### Using ElastiCache With AWS Global Condition Keys

When using AWS Global condition keys that require ElastiCache's Principal, use an OR condition with *both* Principals: `elasticache.amazonaws.com` and `ec.amazonaws.com`.

> **Note**
>
> If you do not add both Principals for ElastiCache, the intended “Allow” or “Deny” action will not be enforced correctly for any resource listed in your policy.

Example of policy with `aws:CalledVia` global condition key:

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:CalledVia": [
            "ec.amazonaws.com",
            "elasticache.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

### Specifying Conditions: Using Condition Keys

To implement fine-grained control, you write an IAM permissions policy that specifies conditions to control a set of individual parameters on certain requests. You then apply the policy to IAM users, groups, or roles that you create using the IAM console.

To apply a condition, you add the condition information to the IAM policy statement. In the following example, you specify the condition that any node-based cluster created will be of node type `cache.r5.large`.

**Note**

- To construct `Condition` elements using condition keys of `String` type, use the case insensitive condition operators `StringEqualsIgnoreCase` or `StringNotEqualsIgnoreCase` to compare a key to a string value.
- ElastiCache processes the input arguments for `CacheNodeType` and `CacheParameterGroupName` in a case insensitive manner. For this reason, the string condition operators `StringEqualsIgnoreCase`, and `StringNotEqualsIgnoreCase` should be used in permissions policies that reference them.

The following shows an example of this permissions policy when using Valkey or Redis OSS.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:CacheNodeType": [
                        "cache.r5.large"
                    ]
                }
            }
        }
    ]
}
```

The following shows an example of this permissions policy when using Memcached.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:CacheNodeType": [
                        "cache.r5.large"
                    ]
                }
            }
        }
    ]
}
```

For more information, see Tagging your ElastiCache resources.

For more information on using policy condition operators, see ElastiCache API permissions: Actions, resources, and conditions reference.

### Example Policies: Using Conditions for Fine-Grained Parameter Control

This section shows example policies for implementing fine-grained access control on the previously listed ElastiCache parameters.

1. **elasticache:MaximumDataStorage**: Specify the maximum data storage of a serverless cache. Using the provided conditions, the customer can not create caches that can store more than a specific amount of data.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDependentResources",
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateServerlessCache"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:serverlesscachesnapshot:*",
                "arn:aws:elasticache:*:*:snapshot:*",
                "arn:aws:elasticache:*:*:usergroup:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateServerlessCache"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:serverlesscache:*"
            ],
            "Condition": {
                "NumericLessThanEquals": {
                    "elasticache:MaximumDataStorage": "30"
                },
                "StringEquals": {
                    "elasticache:DataStorageUnit": "GB"
                }
            }
        }
```

```
    ]
}
```

2. **elasticache:MaximumECPUPerSecond**: Specify the maximum ECPU per second value of a serverless cache. Using the provided conditions, the customer can not create caches that can execute more than a specific number of ECPUs per second.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDependentResources",
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateServerlessCache"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:serverlesscachesnapshot:*",
                "arn:aws:elasticache:*:*:snapshot:*",
                "arn:aws:elasticache:*:*:usergroup:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateServerlessCache"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:serverlesscache:*"
            ],
            "Condition": {
                "NumericLessThanEquals": {
                    "elasticache:MaximumECPUPerSecond": "100000"
                }
            }
        }
    ]
}
```

3. **elasticache:CacheNodeType**: Specify which NodeType(s) a user can create. Using the provided conditions, the customer can specify a single or a range value for a node type.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:CacheNodeType": [
                        "cache.t2.micro",
                        "cache.t2.medium"
                    ]
                }
            }
        }
    ]
}
```

4. **elasticache:CacheNodeType**: With Memcached, specify which NodeType(s) a user can create. Using the provided conditions, the customer can specify a single or a range value for a node type.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:CacheNodeType": [
                        "cache.t2.micro",
                        "cache.t2.medium"
                    ]
                }
            }
        }
    ]
}
```

5. **elasticache:NumNodeGroups**: Create a replication group with fewer than 20 node groups.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
             "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
             "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "NumericLessThanEquals": {
                    "elasticache:NumNodeGroups": "20"
                }
            }
        }
    ]
}
```

6. **elasticache:ReplicasPerNodeGroup**: Specify the replicas per node between 5 and 10.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
```

```
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "NumericGreaterThanEquals": {
                    "elasticache:ReplicasPerNodeGroup": "5"
                },
                "NumericLessThanEquals": {
                    "elasticache:ReplicasPerNodeGroup": "10"
                }
            }
        }
    ]
}
```

7. **elasticache:EngineVersion**: Specify usage of engine version 5.0.6.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
           "Effect": "Allow",
            "Action": [
```

```
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:EngineVersion": "5.0.6"
                }
            }
        }
    ]
}
```

8. **elasticache:EngineVersion**: Specify usage of Memcached engine version 1.6.6

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "StringEquals": {
```

```
                    "elasticache:EngineVersion": "1.6.6"
                }
            }
        }
    ]
}
```

9. **elasticache:EngineType**: Specify using a Valkey or Redis OSS engine only.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:EngineType": "redis"
                }
            }
        }
    ]
```

```
}
```

10 **elasticache:AtRestEncryptionEnabled**: Specify that replication groups would be created only with encryption enabled.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [

         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "Bool": {
                    "elasticache:AtRestEncryptionEnabled": "true"
                }
            }
        }
    ]
}
```

11 **elasticache:TransitEncryptionEnabled**

a. Set the `elasticache:TransitEncryptionEnabled` condition key to `false` for the CreateReplicationGroup action to specify that replication groups can only be created when TLS is not being used:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "Bool": {
                    "elasticache:TransitEncryptionEnabled": "false"
                }
            }
        }
    ]
}
```

When the `elasticache:TransitEncryptionEnabled` condition key is set to `false` in a policy for the CreateReplicationGroup action, a `CreateReplicationGroup` request will be allowed only if TLS is not being used (that is, if the request does not include a `TransitEncryptionEnabled` parameter set to `true` or a `TransitEncryptionMode` parameter set to `required`.

b. Set the `elasticache:TransitEncryptionEnabled` conditon key to `true` for the CreateReplicationGroup action to specify that replication groups can only be created when TLS is being used:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "Bool": {
                    "elasticache:TransitEncryptionEnabled": "true"
                }
            }
        }
    ]
}
```

When the `elasticache:TransitEncryptionEnabled` condition key is set to `true` in a policy for the CreateReplicationGroup action, a `CreateReplicationGroup` request will be allowed only if the request includes a `TransitEncryptionEnabled` parameter set to `true` and a `TransitEncryptionMode` parameter set to `required`.

c. Set `elasticache:TransitEncryptionEnabled` to `true` for the `ModifyReplicationGroup` action to specify that replication groups can only be modified when TLS is being used:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:ModifyReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "BoolIfExists": {
                    "elasticache:TransitEncryptionEnabled": "true"
                }
            }
        }
    ]
}
```

When the `elasticache:TransitEncryptionEnabled` condition key is set to `true` in a policy for the ModifyReplicationGroup action, a `ModifyReplicationGroup` request will be allowed only if the request includes a `TransitEncryptionMode` parameter set to `required`. The `TransitEncryptionEnabled` parameter set to `true` may optionally be included as well, but is not needed in this case to enable TLS.

12 **elasticache:AutomaticFailoverEnabled**: Specify that replication groups would be created only with automatic failover enabled.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
```

```
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "Bool": {
                    "elasticache:AutomaticFailoverEnabled": "true"
                }
            }
        }
    ]
}
```

13 **elasticache:MultiAZEnabled**: Specify that replication groups cannot be created with Multi-AZ disabled.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
```

```
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "Bool": {
                    "elasticache:MultiAZEnabled": "false"
                }
            }
        }
    ]
}
```

14 **elasticache:ClusterModeEnabled**: Specify that replication groups can only be created with cluster mode enabled.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
```

```
                "Bool": {
                    "elasticache:ClusterModeEnabled": "true"
                }
            }
        }
    ]
}
```

15 **elasticache:AuthTokenEnabled**: Specify that replication groups can only be created with AUTH token enabled.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [

         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "Bool": {
                    "elasticache:AuthTokenEnabled": "true"
                }
            }
```

```
        }
    ]
}
```

16 **elasticache:SnapshotRetentionLimit**: Specify the number of days (or min/max) to keep the snapshot. Below policy enforces storing backups for at least 30 days.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [

         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup",
                "elasticache:CreateServerlessCache"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*",
                "arn:aws:elasticache:*:*:serverlesscache:*"
            ],
            "Condition": {
                "NumericGreaterThanEquals": {
                    "elasticache:SnapshotRetentionLimit": "30"
                }
            }
        }
    ]
```

```
}
```

17 **elasticache:KmsKeyId**: Specify usage of customer managed AWS KMS keys.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
        "Sid": "AllowDependentResources",
        "Effect": "Allow",
        "Action": [
            "elasticache:CreateServerlessCache"
        ],
        "Resource": [
            "arn:aws:elasticache:*:*:serverlesscachesnapshot:*",
            "arn:aws:elasticache:*:*:snapshot:*",
            "arn:aws:elasticache:*:*:usergroup:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticache:CreateServerlessCache"
        ],
        "Resource": [
            "arn:aws:elasticache:*:*:serverlesscache:*"
        ],
        "Condition": {
            "StringEquals": {
                "elasticache:KmsKeyId": "my-key"
            }
        }
    }
  ]
}
```

18 **elasticache:CacheParameterGroupName**: Specify a non default parameter group with specific parameters from an organization on your clusters. You could also specify a naming pattern for your parameter groups or block delete on a specific parameter group name. Following is an example constraining usage of only "my-org-param-group".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [

         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:CreateReplicationGroup"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:CacheParameterGroupName": "my-org-param-group"
                }
            }
        }
    ]
}
```

19 **elasticache:CacheParameterGroupName**: With Memcached, specify a non default parameter group with specific parameters from an organization on your clusters. You could also specify a naming pattern for your parameter groups or block delete on a specific parameter group name. Following is an example constraining usage of only "my-org-param-group".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [

         {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "StringEquals": {
                    "elasticache:CacheParameterGroupName": "my-org-param-group"
                }
            }
        }
    ]
}
```

20**elasticache:CreateCacheCluster**: Denying `CreateCacheCluster` action if the request tag `Project` is missing or is not equal to `Dev`, `QA` or `Prod`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
          {
```

```
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:parametergroup:*",
                "arn:aws:elasticache:*:*:subnetgroup:*",
                "arn:aws:elasticache:*:*:securitygroup:*",
                "arn:aws:elasticache:*:*:replicationgroup:*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "elasticache:CreateCacheCluster"
            ],
            "Resource": [
                "arn:aws:elasticache:*:*:cluster:*"
            ],
            "Condition": {
                "Null": {
                    "aws:RequestTag/Project": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticache:CreateCacheCluster",
                "elasticache:AddTagsToResource"
            ],
            "Resource": "arn:aws:elasticache:*:*:cluster:*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Project": [
                        "Dev",
                        "Prod",
                        "QA"
                    ]
                }
            }
        }
    ]
```

```
}
```

21. **elasticache:CacheNodeType**: Allowing `CreateCacheCluster` with `cacheNodeType` cache.r5.large or cache.r6g.4xlarge and tag `Project=XYZ`.

    JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
      "Effect": "Allow",
      "Action": [
        "elasticache:CreateCacheCluster",
        "elasticache:CreateReplicationGroup"
      ],
      "Resource": [
        "arn:aws:elasticache:*:*:parametergroup:*",
        "arn:aws:elasticache:*:*:subnetgroup:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticache:CreateCacheCluster"
      ],
      "Resource": [
        "arn:aws:elasticache:*:*:cluster:*"
      ],
      "Condition": {
        "StringEqualsIfExists": {
          "elasticache:CacheNodeType": [
            "cache.r5.large",
            "cache.r6g.4xlarge"
          ]
        },
        "StringEquals": {
          "aws:RequestTag/Project": "XYZ"
        }
      }
    }
  ]
}
```

22**elasticache:CacheNodeType**: Allowing `CreateCacheCluster` with `cacheNodeType` cache.r5.large or cache.r6g.4xlarge and tag `Project=XYZ`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
      "Effect": "Allow",
      "Action": [
        "elasticache:CreateCacheCluster"
      ],
      "Resource": [
        "arn:aws:elasticache:*:*:parametergroup:*",
        "arn:aws:elasticache:*:*:subnetgroup:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticache:CreateCacheCluster"
      ],
      "Resource": [
        "arn:aws:elasticache:*:*:cluster:*"
      ],
      "Condition": {
        "StringEqualsIfExists": {
          "elasticache:CacheNodeType": [
            "cache.r5.large",
            "cache.r6g.4xlarge"
          ]
        },
        "StringEquals": {
          "aws:RequestTag/Project": "XYZ"
        }
      }
    }
  ]
}
```

23**elasticache:ConnectionType**: Deny creating serverless caches with a public connection type. Using the provided condition, administrators can prevent users from creating caches with public endpoints.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "elasticache:CreateServerlessCache",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "elasticache:ConnectionType": "public"
                }
            }
        }
    ]
}
```

**Note**

When creating polices to enforce tags and other condition keys together, the conditional `IfExists` may be required on condition key elements due to the extra `elasticache:AddTagsToResource` policy requirements for creation requests with the `--tags` parameter.

## Using Service-Linked Roles for Amazon ElastiCache

Amazon ElastiCache uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to an AWS service, such as Amazon ElastiCache. Amazon ElastiCache service-linked roles are predefined by Amazon ElastiCache. They include all the permissions that the service requires to call AWS services on behalf of your clusters.

A service-linked role makes setting up Amazon ElastiCache easier because you don't have to manually add the necessary permissions. The roles already exist within your AWS account but are linked to Amazon ElastiCache use cases and have predefined permissions. Only Amazon

ElastiCache can assume these roles, and only these roles can use the predefined permissions policy. You can delete the roles only after first deleting their related resources. This protects your Amazon ElastiCache resources because you can't inadvertently remove necessary permissions to access the resources.

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.

**Contents**


- Service-Linked Role Permissions for Amazon ElastiCache
  - Permissions to create service-linked role
- Creating a Service-Linked Role (IAM)
  - Creating a Service-Linked Role (IAM Console)
  - Creating a Service-Linked Role (IAM CLI)
  - Creating a Service-Linked Role (IAM API)
- Editing the Description of a Service-Linked Role for Amazon ElastiCache
  - Editing a Service-Linked Role Description (IAM Console)
  - Editing a Service-Linked Role Description (IAM CLI)
  - Editing a Service-Linked Role Description (IAM API)
- Deleting a Service-Linked Role for Amazon ElastiCache
  - Cleaning Up a Service-Linked Role
  - Deleting a Service-Linked Role (IAM Console)
  - Deleting a Service-Linked Role (IAM CLI)
  - Deleting a Service-Linked Role (IAM API)


### Service-Linked Role Permissions for Amazon ElastiCache

#### Permissions to create service-linked role

#### To allow an IAM entity to create AWSServiceRoleForElastiCache service-linked role

Add the following policy statement to the permissions for that IAM entity:

```
{
    "Effect": "Allow",
```

```
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:PutRolePolicy"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/elasticache.amazonaws.com/
AWSServiceRoleForElastiCache*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "elasticache.amazonaws.com"}}
}
```

**To allow an IAM entity to delete AWSServiceRoleForElastiCache service-linked role**

Add the following policy statement to the permissions for that IAM entity:

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/elasticache.amazonaws.com/
AWSServiceRoleForElastiCache*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "elasticache.amazonaws.com"}}
}
```

Alternatively, you can use an AWS managed policy to provide full access to Amazon ElastiCache.

**Creating a Service-Linked Role (IAM)**

You can create a service-linked role using the IAM console, CLI, or API.

**Creating a Service-Linked Role (IAM Console)**

You can use the IAM console to create a service-linked role.

**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 new role**.
3. Under **Select type of trusted entity** choose **AWS Service**.
4. Under **Or select a service to view its use cases**, choose **ElastiCache**.
5. Choose **Next: Permissions**.

6. Under **Policy name**, note that the `ElastiCacheServiceRolePolicy` is required for this role. Choose **Next:Tags**.
7. Note that tags are not supported for Service-Linked roles. Choose **Next:Review**.
8. (Optional) For **Role description**, edit the description for the new service-linked role.
9. Review the role and then choose **Create role**.

#### Creating a Service-Linked Role (IAM CLI)

You can use IAM operations from the AWS Command Line Interface to create a service-linked role. This role can include the trust policy and inline policies that the service needs to assume the role.

#### To create a service-linked role (CLI)

Use the following operation:

```
$ aws iam create-service-linked-role --aws-service-name elasticache.amazonaws.com
```

#### Creating a Service-Linked Role (IAM API)

You can use the IAM API to create a service-linked role. This role can contain the trust policy and inline policies that the service needs to assume the role.

#### To create a service-linked role (API)

Use the CreateServiceLinkedRole API call. In the request, specify a service name of `elasticache.amazonaws.com`.

#### Editing the Description of a Service-Linked Role for Amazon ElastiCache

Amazon ElastiCache does not allow you to edit the AWSServiceRoleForElastiCache 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.

#### Editing a Service-Linked Role Description (IAM Console)

You can use the IAM console to edit a service-linked role description.

#### 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 (IAM CLI)

You can use IAM operations from the AWS Command Line Interface to edit a service-linked role description.

**To change the description of a service-linked role (CLI)**

1. (Optional) To view the current description for a role, use the AWS CLI for IAM operation `get-role`.

   **Example**

   ```
   $ aws iam get-role --role-name AWSServiceRoleForElastiCache
   ```

   Use the role name, not the ARN, to refer to roles with the CLI operations. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, refer to the role as **`myrole`**.
2. To update a service-linked role's description, use the AWS CLI for IAM operation `update-role-description`.

   For Linux, macOS, or Unix:

   ```
   $ aws iam update-role-description \
       --role-name AWSServiceRoleForElastiCache \
       --description "new description"
   ```

   For Windows:

   ```
   $ aws iam update-role-description ^
       --role-name AWSServiceRoleForElastiCache ^
       --description "new description"
   ```

### Editing a Service-Linked Role Description (IAM API)

You can use the IAM API to edit a service-linked role description.

**To change the description of a service-linked role (API)**

1. (Optional) To view the current description for a role, use the IAM API operation GetRole.

   **Example**

   ```
   https://iam.amazonaws.com/
      ?Action=GetRole
      &RoleName=AWSServiceRoleForElastiCache
      &Version=2010-05-08
      &AUTHPARAMS
   ```

2. To update a role's description, use the IAM API operation UpdateRoleDescription.

   **Example**

   ```
   https://iam.amazonaws.com/
      ?Action=UpdateRoleDescription
      &RoleName=AWSServiceRoleForElastiCache
      &Version=2010-05-08
      &Description="New description"
   ```

### Deleting a Service-Linked Role for Amazon ElastiCache

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 is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Amazon ElastiCache does not delete the service-linked role for you.

### Cleaning Up a Service-Linked Role

Before you can use IAM to delete a service-linked role, first confirm that the role has no resources (clusters or replication groups) associated with it.

**To check whether the service-linked role has an active session 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 **Roles**. Then choose the name (not the check box) of the AWSServiceRoleForElastiCache role.

3. On the **Summary** page for the selected role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review recent activity for the service-linked role.

**To delete Amazon ElastiCache resources that require AWSServiceRoleForElastiCache**

- To delete a cluster, see the following:
  - Using the AWS Management Console
  - Using the AWS CLI to delete an ElastiCache cluster
  - Using the ElastiCache API
- To delete a replication group, see the following:
  - Deleting a Replication Group (Console)
  - Deleting a Replication Group (AWS CLI)
  - Deleting a replication group (ElastiCache API)

**Deleting a Service-Linked Role (IAM Console)**

You can use the IAM console to delete a service-linked role.

**To delete 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 select the check box next to the role name that you want to delete, not the name or row itself.
3. For **Role actions** at the top of the page, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. If you want to proceed, choose **Yes, Delete** to submit the service-linked role for deletion.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, after you submit the role for deletion, the deletion task can succeed or fail. If the task fails, you can choose **View details** or **View Resources** from the notifications to learn why the deletion failed.

### Deleting a Service-Linked Role (IAM CLI)

You can use IAM operations from the AWS Command Line Interface to delete a service-linked role.

#### To delete a service-linked role (CLI)

1. If you don't know the name of the service-linked role that you want to delete, enter the following command. This command lists the roles and their Amazon Resource Names (ARNs) in your account.

   ```
   $ aws iam get-role --role-name role-name
   ```

   Use the role name, not the ARN, to refer to roles with the CLI operations. For example, if a role has the ARN `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.

2. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Enter the following to submit a service-linked role deletion request.

   ```
   $ aws iam delete-service-linked-role --role-name role-name
   ```

3. Enter the following to check the status of the deletion task.

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

### Deleting a Service-Linked Role (IAM API)

You can use the IAM API to delete a service-linked role.

#### To delete a service-linked role (API)

1. To submit a deletion request for a service-linked roll, call DeleteServiceLinkedRole. In the request, specify a role name.

Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call GetServiceLinkedRoleDeletionStatus. In the request, specify the `DeletionTaskId`.

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

### ElastiCache API permissions: Actions, resources, and conditions reference

When you set up access control and write permissions policies to attach to an IAM policy (either identity-based or resource-based), use the following table as a reference. The table lists each Amazon ElastiCache API operation and the corresponding actions for which you can grant permissions to perform the action. You specify the actions in the policy's `Action` field, and you specify a resource value in the policy's `Resource` field. Unless indicated otherwise, the resource is required. Some fields include both a required resource and optional resources. When there is no resource ARN, the resource in the policy is a wildcard (*).

You can use condition keys in your ElastiCache policies to express conditions. To see a list of ElastiCache-specific condition keys, along with the actions and resource types to which they apply, see Using condition keys. For a complete list of AWS-wide keys, see AWS global condition context keys in the *IAM User Guide*.

> **Note**
>
> To specify an action, use the `elasticache:` prefix followed by the API operation name (for example, `elasticache:DescribeCacheClusters`).

To see a list of ElastiCache actions, see Actions Defined by Amazon ElastiCache in the *Service Authorization Reference*.

## Compliance validation for Amazon ElastiCache

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs, such as SOC, PCI, FedRAMP, and HIPAA.

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.

- When Amazon ElastiCache is listed as being within scope of a compliance program, this scope applies to all cache engines supported by ElastiCache. This includes ElastiCache for Valkey, ElastiCache for Memcached, and ElastiCache for Redis OSS.

### More information

For general information about AWS Cloud compliance, see the following:

- FIPS Endpoints by Service
- Service updates in ElastiCache
- AWS Cloud Compliance
- Shared Responsibility Model
- AWS PCI DSS Compliance Program

## Resilience in Amazon ElastiCache

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon ElastiCache offers several features to help support your data resiliency and backup needs.

**Topics**

- Mitigating Failures

# Mitigating Failures

When planning your Amazon ElastiCache implementation, you should plan so that failures have a minimal impact upon your application and data. The topics in this section cover approaches you can take to protect your application and data from failures.

**Topics**


- Mitigating Failures when Running Memcached
- Mitigating Failures when Running Valkey or Redis OSS
- Recommendations


## Mitigating Failures when Running Memcached

When running the Memcached engine, you have the following options for minimizing the impact of a failure. There are two types of failures to address in your failure mitigation plans: node failure and Availability Zone failure.

### Mitigating Node Failures

Serverless caches automatically mitigate node failures with a replicated Multi-AZ architecture so that node failures are transparent to your application. To mitigate the impact of a node failure in a node-based cluster, spread your cached data over more nodes. Because node-based clusters do not support replication, a node failure will always result in some data loss from your cluster.

When you create your Memcached cluster you can create it with 1 to 60 nodes, or more by special request. Partitioning your data across a greater number of nodes means you'll lose less data if a node fails. For example, if you partition your data across 10 nodes, any single node stores approximately 10% of your cached data. In this case, a node failure loses approximately 10% of your cache which needs to be replaced when a replacement node is created and provisioned. If the same data were cached in 3 larger nodes, the failure of a node would lose approximately 33% of your cached data.

For information on specifying the number of nodes in a Memcached cluster, see Creating a Memcached cluster (console).

### Mitigating Availability Zone Failures

Serverless caches automatically mitigate availability zone failures with a replicated Multi-AZ architecture so that AZ failures are transparent to your application.

To mitigate the impact of an Availability Zone failure in a node-based cluster, locate your nodes in as many Availability Zones as possible. In the unlikely event of an AZ failure, you will lose the data cached in that AZ, not the data cached in the other AZs.

**Why so many nodes?**

*If my region has only 3 Availability Zones, why do I need more than 3 nodes since if an AZ fails I lose approximately one-third of my data?*

This is an excellent question. Remember that we're attempting to mitigate two distinct types of failures, node and Availability Zone. You're right, if your data is spread across Availability Zones and one of the zones fails, you will lose only the data cached in that AZ, irrespective of the number of nodes you have. However, if a node fails, having more nodes will reduce the proportion of data lost.

There is no "magic formula" for determining how many nodes to have in your cluster. You must weight the impact of data loss vs. the likelihood of a failure vs. cost, and come to your own conclusion.

For information on specifying the number of nodes in a Memcached cluster, see Creating a Memcached cluster (console).

For more information on regions and Availability Zones, see Choosing regions and availability zones for ElastiCache.

## Mitigating Failures when Running Valkey or Redis OSS

When running a Valkey or Redis OSS engine, you have the following options for minimizing the impact of a node or Availability Zone failure.

**Mitigating Node Failures**

Serverless caches automatically mitigate node failures with a Multi-AZ architecture so that node failures are transparent to your application. Node-based clusters must be configured appropriately to mitigate the failure of an individual node.

To mitigate the impact of Valkey or Redis OSS node failures on node-based clusters, you have the following options:

**Topics**

- Mitigating Failures: Valkey or Redis OSS Replication Groups

### Mitigating Failures: Valkey or Redis OSS Replication Groups

A Valkey or Redis OSS replication group is comprised of a single primary node which your application can both read from and write to, and from 1 to 5 read-only replica nodes. Whenever data is written to the primary node it is also asynchronously updated on the read replica nodes.

#### When a read replica fails

1. ElastiCache detects the failed read replica.
2. ElastiCache takes the failed node off line.
3. ElastiCache launches and provisions a replacement node in the same AZ.
4. The new node synchronizes with the primary node.

During this time your application can continue reading and writing using the other nodes.

#### Valkey or Redis OSS Multi-AZ

We recommend enabling Multi-AZ on your Valkey or Redis OSS replication groups for the best availability configuration. Multi-AZ distributes your replicas across Availability Zones and enables automatic failover, so that a primary node failure results in rapid promotion of a replica with minimal write interruption.

#### When Multi-AZ is enabled

1. ElastiCache detects the primary node failure.
2. ElastiCache automatically promotes the read replica node with the least replication lag to primary node.
3. The other replicas sync with the new primary node.
4. ElastiCache spins up a read replica in the failed primary's AZ.
5. The new node syncs with the newly promoted primary.

Failing over to a replica node is generally faster than creating and provisioning a new primary node. This means your application can resume writing to your primary node sooner than if Multi-AZ were not enabled.

For more information about how Multi-AZ and automatic failover work together, see Automatic failover and Multi-AZ.

### When Multi-AZ is not enabled

1. ElastiCache detects primary failure.
2. ElastiCache takes the primary offline.
3. ElastiCache creates and provisions a new primary node to replace the failed primary.
4. If replicas exist, ElastiCache syncs the new primary with one of the existing replicas. If no replicas exist, the new primary starts empty.
5. The new node functions as the cluster's primary node.

> **Note**
>
> While it is possible to have automatic failover enabled without Multi-AZ, replicas are not guaranteed to be in different Availability Zones in this configuration. Failover still promotes a replica to primary if one is available, but this does not protect against AZ-level failures. For most use cases, we recommend enabling Multi-AZ.

During steps 1 through 4 of this process, your application can't write to the primary node. However, if you have replica nodes, your application can continue reading from them during this process.

For added protection, we recommend that you launch the nodes in your replication group in different Availability Zones (AZs) by enabling Multi-AZ. If you do this, an AZ failure will only impact the nodes in that AZ and not the others.

For more information, see High availability using replication groups.

### Mitigating Availability Zone Failures

Serverless caches automatically mitigate availability zone failures with a replicated Multi-AZ architecture so that AZ failures are transparent to your application.

To mitigate the impact of an Availability Zone failure in a node-based cluster, locate your nodes for each shard in as many Availability Zones as possible.

No matter how many nodes you have in a shard, if they are all located in the same Availability Zone, a catastrophic failure of that AZ results in your losing all your shard's data. However, if you locate your nodes in multiple AZs, a failure of any AZ results in your losing only the nodes in that AZ.

Any time you lose a node you can experience a performance degradation since read operations are now shared by fewer nodes. This performance degradation will continue until the nodes are replaced.

For information on specifying the Availability Zones for Valkey or Redis OSS nodes, see Creating a Valkey (cluster mode disabled) cluster (Console).

For more information on regions and Availability Zones, see Choosing regions and availability zones for ElastiCache.

## Recommendations

We recommend creating serverless caches over node-based clusters, as you automatically obtain better fault tolerance without additional configuration. When creating a node-based cluster, however, there are two types of failures you need to plan for: individual node failures and broad Availability Zone failures. The best failure mitigation plan will address both kinds of failures.

### Minimizing the Impact of Node Failures

To minimize the impact of a node failure when using Valkey or Redis OSS, we recommend that your implementation use multiple nodes in each shard and distribute the nodes across multiple Availability Zones. This is done automatically for serverless caches.

For node-based clusters on Valkey or Redis OSS, we recommend that you enable Multi-AZ on your replication group so that ElastiCache will automatically fail over to a replica if the primary node fails.

When running Memcached and partitioning your data across nodes, the more nodes you use the smaller the data loss if any one node fails.

### Minimizing the Impact of Availability Zone Failures

To minimize the impact of an Availability Zone failure, we recommend launching your nodes in as many different Availability Zones as are available. Spreading your nodes evenly across AZs will minimize the impact in the unlikely event of an AZ failure. This is done automatically for serverless caches.

### Other precautions

If you're running Valkey or Redis OSS, then in addition to the above, we recommend that you schedule regular backups of your cluster. Backups (snapshots) create a .rdb file you can use to restore your cache in case of failure or corruption. For more information, see Snapshot and restore.

# Infrastructure security in AWS ElastiCache

As a managed service, AWS ElastiCache is protected by the AWS global network security procedures that are described in the Security and Compliance section at AWS Architecture Center.

You use AWS published API calls to access ElastiCache through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. We recommend TLS 1.3 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

# Service updates in ElastiCache

ElastiCache automatically monitors your fleet of caches, clusters, and nodes to apply service updates as they become available. Service updates for serverless caches are automatically and transparently applied. For node-based clusters, you set up a predefined maintenance window. Critical security updates will automatically be applied in these maintenance windows. All other updates will be available via a self-service update.

With self-service updates, you control when and which updates are applied to your node-based clusters. You can also monitor the progress of these updates to your selected ElastiCache cluster in real time.

**Topics**

- Managing service updates for node-based clusters

## Managing service updates for node-based clusters

ElastiCache service updates for node-based clusters are released on a regular basis. If you have one or more qualifying node-based clusters for those service updates, you receive notifications through email, SNS, the Personal Health Dashboard (PHD), and Amazon CloudWatch events when the updates are released. The updates are also displayed on the **Service Updates** page on the ElastiCache console. By using this dashboard, you can view all the service updates and their status

for your ElastiCache fleet. Service updates for serverless caches are transparently applied and cannot be managed via **Service Updates**.

You control when to apply an update before an auto-update starts. There are three types of service updates:

- **security-update**: Includes the latest security patches. We strongly recommend that you apply security updates as soon as possible to ensure that your ElastiCache clusters are always up-to-date with the latest security patches.
- **engine-update**: Includes patches or minor engine version updates related to performance or stability optimizations for your current engine version.
- **engine-major-version-update**: Includes a major version or engine type change, typically due to your current engine version reaching its end of life. We recommend that you carefully review the version compatibility notes from the Engine versions and upgrading in ElastiCache before applying such updates.

The following sections explore these options in detail.

## Applying the service updates

You can start applying the service updates to your fleet from the time that the updates have an **available** status. Service updates are cumulative. In other words, any updates that you haven't applied yet are included with your latest update.

If a service update has auto-update enabled, you can choose to note any action when it becomes available. ElastiCache will schedule to apply the update during one of your clusters' upcoming maintenance windows after the **Auto-update start date**. You will receive related notifications for each stage of the update.

> **Note**
>
> You can apply only those service updates that have an **available** or **scheduled** status.

For more information about reviewing and applying any service-specific updates to applicable ElastiCache clusters, see Applying the service updates using the console.

When a new service update is available for one or more of your ElastiCache clusters, you can use the ElastiCache console, API, or AWS CLI to apply the update. The following sections explain the options that you can use to apply updates.

**Applying the service updates using the console**

To view the list of available service updates, along with other information, go to the **Service Updates** page in the console.

1. Sign in to the AWS Management Console and open the Amazon ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. On the navigation pane, choose **Service Updates**.
3. Under **Service updates** you can view the following:

   - **Service update name**: The unique name of the service update
   - **Update type**: The type of the service update, which is one of **security-update**, **engine-update** or **engine-major-version-update**
   - **Update severity**: The priority of applying the update:
     - **critical:** We recommend that you apply this update immediately (within 14 days or less).
     - **important:** We recommend that you apply this update as soon as your business flow allows (within 30 days or less).
     - **medium:** We recommend that you apply this update as soon as you can (within 60 days or less).
     - **low:** We recommend that you apply this update as soon as you can (within 90 days or less).
   - **Engine version**: If the update type is engine-update, the engine version that is being updated.
   - **Release Date**: When the update is released and available to apply on your clusters.
   - **Recommended Apply By Date**: ElastiCache guidance date to apply the updates by.
   - **Auto-update after due date**: Indicates whether ElastiCache automatically schedules the service update to apply during your cluster's maintenance window after the Recommended Apply By Date passes. ElastiCache sets this value based on the criticality of the patch. You cannot modify this value. If the value is `Yes`, ElastiCache schedules the update automatically. If the value is `No`, ElastiCache does not automatically apply the update after the due date, and you can manually apply it or choose not to apply it.

- **Status**: The status of the update, which is one of the following:
  - **available:** The update is available for requisite clusters.
  - **complete:** The update has been applied.
  - **cancelled:** The update has been canceled and is no longer necessary.
  - **expired:** The update is no longer available to apply.

4. Choose an individual update (not the button to its left) to view details of the service update.

   In the **Cluster update status** section, you can view a list of clusters where the service update has not been applied or has just been applied recently. For each cluster, you can view the following:

   - **Cluster name**: The name of the cluster
   - **Nodes updated:** The ratio of individual nodes within a specific cluster that were updated or remain available for the specific service update.
   - **Update Type**: The type of the service update, which is one of **security-update** or **engine-update**
   - **Status**: The status of the service update on the cluster, which is one of the following:
     - *available*: The update is available for the requisite cluster.
     - *in-progress*: The update is being applied to this cluster.
     - *scheduled*: The update date has been scheduled.
     - *complete*: The update has been successfully applied. Cluster with a complete status will be displayed for 7 days after its completion.

   If you chose any or all of the clusters with the **available** or **scheduled** status, and then chose **Apply now**, the update will start being applied on those clusters.

### Applying the service updates using the AWS CLI

After you receive notification that service updates are available, you can inspect and apply them using the AWS CLI:

- To retrieve a description of the service updates that are available, run the following command:

```
aws elasticache describe-service-updates --service-update-status available
```

For more information, see describe-service-updates.

- To apply a service update on a list of replication groups, run the following command:

```
aws elasticache batch-apply-update-action \
  --service-update-name my-service-update \
  --replication-group-ids my-replication-group-1 my-replication-group-2
```

For a cache cluster, run the following command:

```
aws elasticache batch-apply-update-action \
  --service-update-name my-service-update \
  --cache-cluster-ids my-cache-cluster-1 my-cache-cluster-2
```

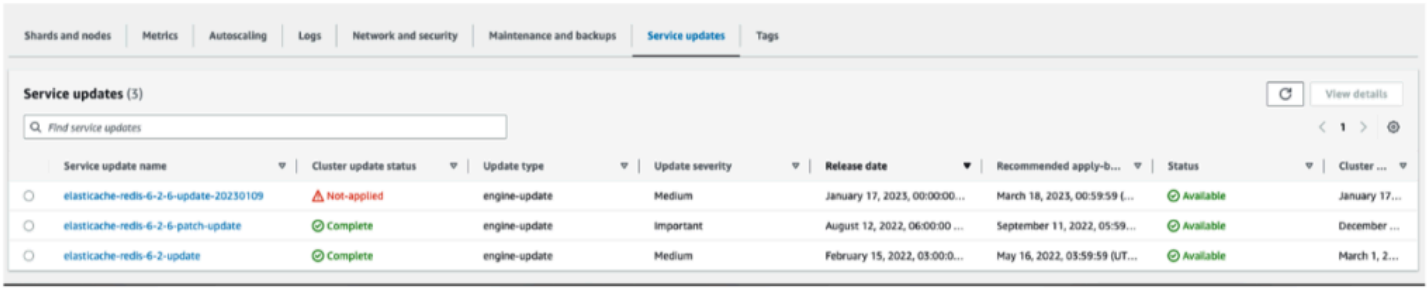
For more information, see batch-apply-update-action.

## Verifying you have the latest Service Update Applied using the AWS console

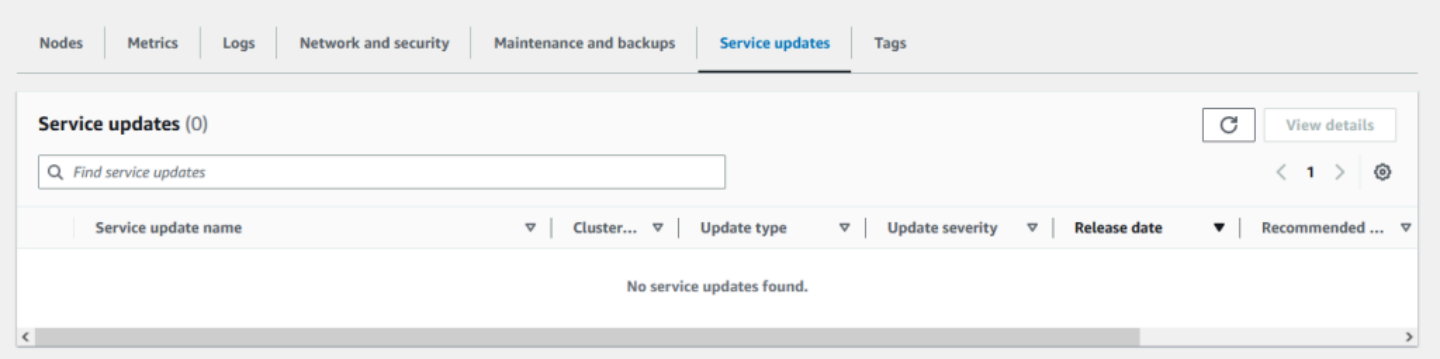
You can verify your ElastiCache for Valkey or Redis OSS clusters are running the latest service update by following these steps:

1. Choose an applicable cluster on the **Valkey or Redis OSS caches** page
2. Choose **Service updates** in the navigation pane to see the applicable service updates for that cluster, if any.

If the console displays a list of service updates, you can select the service update and choose **Apply now**.




If the console displays “No service updates found”, it means the ElastiCache for Valkey or Redis OSS cluster already has the latest service update applied.




## Stopping the service updates

You can stop updates to clusters if needed. For example, you might want to stop updates if you have an unexpected surge to your clusters that are undergoing updates. Or you might want to stop updates if they're taking too long and interrupting your business flow at a peak time.

The Stopping operation immediately interrupts all updates to those clusters and any nodes that are yet to be updated. It continues to completion any nodes that have an **in progress** status. However, it ceases updates to other nodes in the same cluster that have an **update available** status and reverts them to a **Stopping** status.

When the **Stopping** workflow is complete, the nodes that have a **Stopping** status change to a **Stopped** status. Depending on the workflow of the update, some clusters won't have any nodes updated. Other clusters might include some nodes that are updated and others that still have an **update available** status.

You can return later to finish the update process as your business flows permit. In this case, choose the applicable clusters that you want to complete updates on, and then choose **Apply Now**. For more information, see Applying the service updates.

### Using the console

You can interrupt a service update using the ElastiCache console. The following demonstrates how to do this:

- After a service update has progressed on a selected cluster, the ElastiCache console displays the **View/Stop Update** tab at the top of the ElastiCache dashboard.
- To interrupt the update, choose **Stop Update**.
- When you stop the update, choose the cluster and examine the status. It reverts to a **Stopping** status and eventually a **Stopped** status.

### Using the AWS CLI

You can interrupt a service update using the AWS CLI. The following code example shows how to do this.

For a replication group, do the following:

```
aws elasticache batch-stop-update-action \
  --service-update-name my-service-update \
  --replication-group-ids my-replication-group-1 my-replication-group-2
```

For a cache cluster, do the following:

```
aws elasticache batch-stop-update-action \
  --service-update-name my-service-update \
  --cache-cluster-ids my-cache-cluster-1 my-cache-cluster-2
```

For more information, see BatchStopUpdateAction.

# Common Vulnerabilities and Exposures (CVE): Security vulnerabilities addressed in ElastiCache

Common Vulnerabilities and Exposures (CVE) is a list of entries for publicly known cybersecurity vulnerabilities. Each entry is a link that contains an identification number, a description, and at least one public reference. You can find on this page a list of security vulnerabilities that have been addressed in ElastiCache, as well as CVEs that do not affect ElastiCache.

We recommend that you always upgrade to the latest ElastiCache Valkey, Redis OSS or ElastiCache Memcached versions to be protected against known vulnerabilities. When operating an ElastiCache Serverless Cache, CVE fixes are automatically applied to your cache. When operating node-based clusters with Valkey or Redis OSS, ElastiCache exposes the PATCH component. For example, when using ElastiCache for Redis OSS version 6.2.6, the major version is 6, the minor version is 2, and the patch version is 6. PATCH versions are for backwards-compatible bug fixes, security fixes, and non-functional changes.

## CVEs addressed in Amazon ElastiCache

The following table lists Common Vulnerabilities and Exposures (CVEs) and the ElastiCache engine versions that address them. In the table, Yes means that the engine version addresses the CVE,

and Not applicable means that the CVE does not affect that engine version. If a vulnerability does not have a CVE ID, the table lists its GitHub Security Advisory (GHSA) ID, which links to the advisory on the GitHub website. If your ElastiCache Valkey or Redis OSS cluster is running a version without the security fix, you can either upgrade to a more recent ElastiCache Valkey or Redis OSS version containing the fix, or if you are on a version containing the fix, ensure you have the latest service update applied by referring to Managing service updates for node-based clusters. For more information about the supported ElastiCache engine versions and how to upgrade, see Engine versions and upgrading in ElastiCache.

**Note**

An asterisk (*) in the following table indicates you must have the latest service update applied for the cluster running the version specified in order to address the security vulnerability. For more information about how to verify you have the latest service update applied for the version your cluster is running on, see Managing service updates for node-based clusters.

| CVE | Valkey 9.1 | Valkey 9.0 | Valkey 8.2 | Valkey 8.1 | Valkey 8.0 | Valkey 7.2 | Redis OSS 7.1 | Redis OSS 7.0 | Redis OSS 6.2 | Redis OSS 6.0 | Redis OSS 5.0 | Redis OSS 4.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GHSA-f4q9-j22m-xqq9 on the GitHub website* | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2026-66373* | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable |
| CVE-2026-25589 | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable |
| CVE-2026-25588 | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable |

| CVE | Valkey 9.1 | Valkey 9.0 | Valkey 8.2 | Valkey 8.1 | Valkey 8.0 | Valkey 7.2 | Redis OSS 7.1 | Redis OSS 7.0 | Redis OSS 6.2 | Redis OSS 6.0 | Redis OSS 5.0 | Redis OSS 4.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CVE-2026-25243* | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable |
| CVE-2026-23631 | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable |
| CVE-2026-23479* | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable |
| CVE-2025-67733* | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable | Not applicable |
| CVE-2025-49844* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2025-49819* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2025-48367* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2025-46844* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2025-46818* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |

| CVE | Valkey 9.1 | Valkey 9.0 | Valkey 8.2 | Valkey 8.1 | Valkey 8.0 | Valkey 7.2 | Redis OSS 7.1 | Redis OSS 7.0 | Redis OSS 6.2 | Redis OSS 6.0 | Redis OSS 5.0 | Redis OSS 4.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CVE-2025-46817* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2025-32023* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2025-27151* | Not applicable | Not applicable | Not applicable | Yes | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable |
| CVE-2025-21605* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2024-46981 | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2024-31449* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2024-31228* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2024-31227* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2023-41056* | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable |

| CVE | Valkey 9.1 | Valkey 9.0 | Valkey 8.2 | Valkey 8.1 | Valkey 8.0 | Valkey 7.2 | Redis OSS 7.1 | Redis OSS 7.0 | Redis OSS 6.2 | Redis OSS 6.0 | Redis OSS 5.0 | Redis OSS 4.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CVE-2023-28425* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Not applicable | Not applicable | Not applicable | Not applicable |
| CVE-2023-25155 | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2023-22458 | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2022-36021* | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2022-35977* | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2022-35951* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Not applicable | Not applicable | Not applicable | Not applicable |
| CVE-2022-31144* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Not applicable | Not applicable | Not applicable | Not applicable |
| CVE-2022-24834* | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CVE-2021-41099* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |

| CVE | Valkey 9.1 | Valkey 9.0 | Valkey 8.2 | Valkey 8.1 | Valkey 8.0 | Valkey 7.2 | Redis OSS 7.1 | Redis OSS 7.0 | Redis OSS 6.2 | Redis OSS 6.0 | Redis OSS 5.0 | Redis OSS 4.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CVE-2021-32762* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32761* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32687* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32675* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32672* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32628* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32627* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32626* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-32625* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Not applicable | Not applicable |

| CVE | Valkey 9.1 | Valkey 9.0 | Valkey 8.2 | Valkey 8.1 | Valkey 8.0 | Valkey 7.2 | Redis OSS 7.1 | Redis OSS 7.0 | Redis OSS 6.2 | Redis OSS 6.0 | Redis OSS 5.0 | Redis OSS 4.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CVE-2021-29478* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-29477* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Not applicable | Not applicable |
| CVE-2021-21309* | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Not applicable | Yes | Yes | Yes | Yes | Not applicable | Not applicable |

## CVEs that do not affect Amazon ElastiCache

The following CVEs do not affect Amazon ElastiCache for Valkey or Redis OSS.

- CVE-2026-21864
- CVE-2026-21863
- CVE-2024-51741
- CVE-2023-45145
- CVE-2023-28856
- CVE-2022-24736
- CVE-2022-24735

# Logging and monitoring in Amazon ElastiCache

To manage your cache, it's important that you know how your caches are performing. ElastiCache generates metrics that are published to Amazon CloudWatch Logs for monitoring your cache performance. In addition, ElastiCache generates events when significant changes happen on your cache resources (e.g. a new cache is created, or a cache is deleted).

**Topics**


- Metrics and events for Valkey and Redis OSS serverless caches
- Metrics and events for node-based Valkey and Redis OSS clusters
- Monitoring ElastiCache with OpenTelemetry metrics
- Metrics and events for Memcached caches and clusters
- Logging Amazon ElastiCache API calls with AWS CloudTrail
- Amazon SNS monitoring of ElastiCache events
- Log delivery
- Monitoring use with CloudWatch Metrics


## Metrics and events for Valkey and Redis OSS serverless caches

ElastiCache offers a wide variety of metrics and events for monitoring when working with serverless caches. This includes CloudWatch metrics, command level metrics, and event logs which can be ingested via Amazon EventBridge.

> **Note**
>
> Amazon ElastiCache offers both serverless caches and node-based clusters. This page covers metrics and events for *serverless caches*. If you are using node-based clusters, see Metrics and events for node-based Valkey and Redis OSS clusters.

**Topics**


- Serverless cache metrics
- Serverless cache events

# Serverless cache metrics

**Dimension names for serverless caches**

Serverless caches use `clusterId` as the CloudWatch dimension name, which corresponds to your serverless cache name. This differs from node-based clusters, which use `CacheClusterId` as the dimension name.

The `AWS/ElastiCache` namespace includes the following CloudWatch metrics for your Valkey or Redis OSS serverless caches.

*Metric codes for Valkey or Redis OSS*

| Metric | Description | Unit |
|---|---|---|
| BytesUsedForCache | The total number of bytes used by the data stored in your cache, including search indexes. | Bytes |
| ElastiCacheProcessingUnits | The total number of ElastiCacheProcessingUnits (ECPUs) consumed by the requests executed on your cache | Count |
| SearchNumberOfIndexes | Number of search indexes. | Count |
| SearchTotalIndexed Documents | Total number of keys in all search indexes. | Count |
| SearchBytesUsedForCache | The total number of bytes that search indexes add to the data stored in your cache. | Bytes |
| SuccessfulReadRequestLatency | Latency of successful read requests. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| SuccessfulWriteRequestLatency | Latency of successful write requests | Microseconds |
| TotalCmdsCount | Total count of all commands executed on your cache | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CacheHitRate | The hit rate of your cache, expressed as a ratio from 0 to 1. This is calculated as `CacheHits / (CacheHits + CacheMisses)` for each one-minute interval. For example, a value of 0.8 means that 80 percent of read-only key lookups were hits.<br><br>When you use the Average statistic over periods longer than one minute, the result is an average of the per-minute ratios. To get the exact hit rate over a longer period, use CloudWatch metric math: `CacheHits / (CacheHits + CacheMisses)` , with the Sum statistic for both metrics. To view it as a percentage, multiply by 100.<br><br>On node-based clusters, `CacheHitRate` is reported as a percentage from 0 to 100. If you reuse dashboards or alarms from a node-based cluster, divide their `CacheHitRate` thresholds by 100. | None |
| CacheHits | The number of successful read-only key lookups in the cache. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CacheMisses | The number of unsuccessful read-only key lookups in the cache. | Count |
| CurrConnections | The number of client connections to your cache. | Count |
| ThrottledCmds | The number of requests that were throttled by ElastiCac he because the workload was scaling faster than ElastiCac he can scale. | Count |
| NewConnections | The total number of connections that have been accepted by the server during this period. | Count |
| CurrItems | The number of items in the cache. | Count |
| CurrVolatileItems | The number of items in the cache with TTL. | Count |
| NetworkBytesIn | Total bytes transferred in to cache | Bytes |
| NetworkBytesOut | Total bytes transferred out of cache | Bytes |
| Evictions | The count of keys evicted by the cache | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| IamAuthenticationExpirations | The total number of expired IAM-authenticated Valkey or Redis OSS connections. You can find more information about Authenticating with IAM in ElastiCache in the user guide. | Count |
| IamAuthenticationThrottling | The total number of throttled IAM-authenticated Valkey or Redis OSS AUTH or HELLO requests. You can find more information about Authenticating with IAM in ElastiCache in the user guide. | Count |
| KeyAuthorizationFailures | The total number of failed attempts by users to access keys they don't have permission to access. We suggest setting an alarm on this to detect unauthorized access attempts. | Count |
| AuthenticationFailures | The total number of failed attempts to authenticate to Valkey or Redis OSS using the AUTH command. We suggest setting an alarm on this to detect unauthorized access attempts. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CommandAuthorizationFailures | The total number of failed attempts by users to run commands they don't have permission to call. We suggest setting an alarm on this to detect unauthorized access attempts. | Count |

### Command level metrics

ElastiCache also emits the following command level metrics. For each command type, ElastiCache emits the total count of commands and the number of ECPUs consumed by that command type.

| Metric | Description | Unit |
| --- | --- | --- |
| EvalBasedCmds | The number of get commands the cache has received. | Count |
| EvalBasedCmdsECPUs | ECPUs consumed by eval-based commands. | Count |
| GeoSpatialBasedCmds | The total number of commands for geospatial-based commands. This is derived from the Valkey or Redis OSS commandstats statistic. It's derived by summing all of the geo type of commands: geoadd, geodist, geohash, geopos, georadius, and georadius bymember. | Count |
| GeoSpatialBasedCmdsECPUs | ECPUs consumed by geospatial-based commands. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| GetTypeCmds | The total number of read-only type commands. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the read-only type commands (get, hget, scard, lrange, and so on.) | Count |
| GetTypeCmdsECPUs | ECPUs consumed by read commands. | Count |
| HashBasedCmds | The total number of commands that are hash-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more hashes (hget, hkeys, hvals, hdel, and so on). | Count |
| HashBasedCmdsECPUs | ECPUs consumed by hash-based commands. | Count |
| HyperLogLogBasedCmds | The total number of HyperLogLog-based commands. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the pf type of commands (pfadd, pfcount, pfmerge, and so on.). | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| HyperLogLogBasedCmdsECPUs | ECPUs consumed by HyperLogLog-based commands. | Count |
| JsonBasedCmds | The total number of JSON commands, including both read and write commands. This is derived from the Valkey or Redis OSS commandstats statistic by summing all JSON commands that act upon JSON keys. | Count |
| JsonBasedCmdsECPUs | ECPUs consumed by all JSON commands, including both read and write commands. | Count |
| JsonBasedGetCmds | The total number of JSON read-only commands. This is derived from the Valkey or Redis OSS commandstats statistic by summing all JSON read commands that act upon JSON keys. | Count |
| JsonBasedGetCmdsECPUs | ECPUs consumed by JSON read-only commands. | Count |
| JsonBasedSetCmds | The total number of JSON write commands. This is derived from the Valkey or Redis OSS commandstats statistic by summing all JSON write commands that act upon JSON keys. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| JsonBasedSetCmdsECPUs | ECPUs consumed by JSON write commands. | Count |
| SearchBasedCmds | Total number of search commands. | Count |
| SearchBasedCmdsECPUs | ECPUs consumed by search commands. | Count |
| KeyBasedCmds | The total number of commands that are key-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more keys across multiple data structures (del, expire, rename, and so on.). | Count |
| KeyBasedCmdsECPUs | ECPUs consumed by key-based commands. | Count |
| ListBasedCmds | The total number of commands that are list-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more lists (lindex, lrange, lpush, ltrim, and so on). | Count |
| ListBasedCmdsECPUs | ECPUs consumed by list-base d commands. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| NonKeyTypeCmds | The total number of commands that are not key-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that do not act upon a key, for example, acl, dbsize or info. | Count |
| NonKeyTypeCmdsECPUs | ECPUs consumed by non-key-based commands. | Count |
| PubSubBasedCmds | The total number of commands for pub/sub functionality. This is derived from the Valkey or Redis OSS commandstats statistic s by summing all of the commands used for pub/ sub functionality: publish, pubsub, ssubscribe, sunsubscr ibe, spublish, subscribe, and unsubscribe. Note that pattern-based commands (psubscribe, punsubscribe) are not supported on serverless caches. | Count |
| PubSubBasedCmdsECPUs | ECPUs consumed by pub/sub-based commands. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| SetBasedCmds | The total number of commands that are set-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more sets (scard, sdiff, sadd, sunion, and so on). | Count |
| SetBasedCmdsECPUs | ECPUs consumed by set-based commands. | Count |
| SetTypeCmds | The total number of write types of commands. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the mutative types of commands that operate on data (set, hset, sadd, lpop, and so on.) | Count |
| SetTypeCmdsECPUs | ECPUs consumed by write commands. | Count |
| SortedSetBasedCmds | The total number of commands that are sorted set-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more sorted sets (zcount, zrange, zrank, zadd, and so on). | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| SortedSetBasedCmdsECPUs | ECPUs consumed by sorted-based commands. | Count |
| StringBasedCmds | The total number of commands that are string-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more strings (strlen, setex, setrange, and so on). | Count |
| StringBasedCmdsECPUs | ECPUs consumed by string-based commands. | Count |
| StreamBasedCmds | The total number of commands that are stream-based. This is derived from the Valkey or Redis OSS commandstats statistic by summing all of the commands that act upon one or more streams data types (xrange, xlen, xadd, xdel, and so on). | Count |
| StreamBasedCmdsECPUs | ECPUs consumed by stream-based commands. | Count |

## Serverless cache events

ElastiCache logs events that relate to your serverless cache. This information includes the date and time of the event, the source name and source type of the event, and a description of the event.

You can easily retrieve events from the log using the ElastiCache console, the AWS CLI describe-events command, or the ElastiCache API action `DescribeEvents`.

You can choose to monitor, ingest, transform, and act on ElastiCache events using Amazon EventBridge. Learn more in the Amazon EventBridge getting started guide.

### Viewing ElastiCache events (Console)

To view events using the ElastiCache console:

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/
2. To see a list of all available events, in the navigation pane, choose **Events**.
3. On the *Events* screen each row of the list represents one event and displays the event source, the event type, the GMT time of the event, and a description of the event. Using the **Filter** you can specify whether you want to see all events, or just events of a specific type in the event list.

### Viewing ElastiCache events (AWS CLI)

To generate a list of ElastiCache events using the AWS CLI, use the command describe-events. You can use optional parameters to control the type of events listed, the time frame of the events listed, the maximum number of events to list, and more.

The following code lists up to 40 serverless cache events.

```
aws elasticache describe-events --source-type serverless-cache --max-items 40
```

The following code lists all events for serverless caches for the past 24 hours (1440 minutes).

```
aws elasticache describe-events --source-type serverless-cache --duration 1440
```

### Serverless Events

This section documents the different types of events that you may receive for your serverless caches.

### Serverless Cache Creation Events

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| Cache created | Cache arn | creation | serverless-cache | Cache <cache-name> is created and ready to use. |
| Cache created | Cache arn<br>Snapshot arn | creation | serverless-cache | Cache <cache-name> is created and data was restored from snapshot. Your cache is ready to use. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Insufficient free IP addresses to create VPC endpoint. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Invalid subnets provided in the request. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Quota limit reached for creating VPC endpoint. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-na |

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| | | | | me>. You do not have permissio ns to create a VPC endpoint. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-na me>. A user with an incompatible Valkey or Redis OSS version is present in user group <user-gro up-name>. |
| Cache creation failed | Cache arn<br><br>Cache snapshot arn | failure | serverless-cache | Failed to create cache <cache-name>. The provided user group <user-gro up-name> does not exist. |

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Data restoration from snapshot failed because <reason>.<br><br>Failure reasons:<br><br>• failed to retrieve file from S3.<br>• expected md5 does not match actual md5.<br>• the provided RDB file has an unsupport ed version. |

**Serverless Cache Update Events (Valkey or Redis OSS)**

| Detail-Type | Resources list | Category | Source | Message |
|---|---|---|---|---|
| Cache updated | Cache arn | configuration change | serverless-cache | SecurityGroups updated for the cache <cache-na me>. |
| Cache updated | Cache arn | configuration change | serverless-cache | Tags updated for the cache <cache-name>. |

| Detail-Type | Resources list | Category | Source | Message |
|---|---|---|---|---|
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. A user with an incompatible Valkey or Redis OSS version is present in user group <user-group-name>. |
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. SecurityGroups update failed. |
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. SecurityGroups update failed because of insufficient permissions. |
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. SecurityGroups update failed because the SecurityGroups are invalid. |

### Serverless Cache Deletion Events (Valkey or Redis OSS)

| Detail-Type | Resources list | Category | Source | Message |
|---|---|---|---|---|
| Cache deleted | Cache arn | deletion | serverless-cache | Cache <cache-name> was deleted. |

### Serverless Cache Usage Limit Events (Valkey or Redis OSS)

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| Cache updated | Cache arn | configuration change | serverless-cache | Limits updated for the cache <cache-name>. |
| Cache limit approaching | Cache arn | notification | serverless-cache | Slot <X> is using more than <Y> % of the per-slot limit of 32 GB. For example, Slot 10 is using more than 90% of the per-slot limit of 32 GB. |
| Cache updated failed | Cache arn | failure | serverless-cache | A limit update to the cache <cache-name> failed because the cache was deleted. |
| Cache updated failed | Cache arn | failure | serverless-cache | A limit update to the cache <cache-name> failed due to |

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| | | | | invalid configuration. |
| Cache updated failed | Cache arn | failure | serverless-cache | A limit update to the cache <cache-name> failed because current cached data exceeds new limits. Please flush some data before applying the limits. |

**Serverless Cache Snapshot Events (Valkey or Redis OSS)**

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot created | Cache arn<br>Snapshot arn | creation | serverless-cache-snapshot | Snapshot <snapshot-name> created for cache <cache-name>. |
| Snapshot creation failed | Cache arn<br>Snapshot arn | failure | serverless-cache-snapshot | Failed to create snapshot for cache <cache-name>. Snapshot <snapshot-name> creation failed with Customer Managed |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| | | | | Key <key-id> <reason>.<br><br>Failure reason messages:<br><br>• because Customer Managed Key is disabled<br>• because Customer Managed Key cannot be found<br>• because the request timed out |
| Snapshot creation failed | Cache arn<br><br>Snapshot arn | failure | serverless-cache-snapshot | Failed to create snapshot for cache <cache-name>. Snapshot <snapshot-name> creation failed because <reason>.<br><br>Default reason:<br><br>• because of an internal error |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket %s because ElastiCache does not have permissions to the bucket. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because there is already an object with the same name in the bucket. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because bucket owner account Id has changed. |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because the S3 bucket is not accessible. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because the bucket is not accessible. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because bucket does not exist. |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' with source snapshot Customer Managed Key %s <reason>. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s'. |
| Snapshot copy failed | Snapshot arn-1<br>Snapshot arn-2 | failure | serverless-cache-snapshot | Failed to copy snapshot <snapshot-name>. Could not copy snapshot '%s' to snapshot '%s' with source snapshot Customer Managed Key <key-id> <reason-name>. |

| Detail-Type | Resources-list | Category | Source | Message |
| --- | --- | --- | --- | --- |
| Snapshot copy failed | Snapshot arn-1<br>Snapshot arn-2 | failure | serverless-cache-snapshot | Failed to copy snapshot <snapshot-name>. Could not copy snapshot '%s' to snapshot '%s' with target snapshot Customer Managed Key '%s' '%s'. |

# Metrics and events for node-based Valkey and Redis OSS clusters

ElastiCache offers a wide variety of metrics and events for monitoring node-based Valkey and Redis OSS clusters. This includes host-level metrics, command level metrics, and event logs available via the AWS CLI and the Amazon Simple Notification Service (SNS).

> **Note**
>
> Amazon ElastiCache offers both serverless caches and node-based clusters. This page covers metrics and events for *node-based clusters*. If you are using serverless caches, see Metrics and events for Valkey and Redis OSS serverless caches.

**Topics**


- OpenTelemetry metrics for node-based Valkey clusters (recommended)
- Classic CloudWatch metrics for Valkey and Redis OSS node-based clusters
- Events for node-based clusters (Valkey and Redis OSS)


## OpenTelemetry metrics for node-based Valkey clusters (recommended)

ElastiCache emits OpenTelemetry (OTLP) metrics for node-based Valkey replication groups to CloudWatch. These metrics provide a broad set of metrics from the cache engine and the host, rich attributes for filtering and aggregation, and support for querying with PromQL.

OpenTelemetry metrics are available for node-based Valkey replication groups running any Valkey version. Serverless caches, Redis OSS, Memcached, and AWS Outposts are not supported.

For more information, see Monitoring ElastiCache with OpenTelemetry metrics and OpenTelemetry metrics reference for Amazon ElastiCache.

## Classic CloudWatch metrics for Valkey and Redis OSS node-based clusters

When you create a node-based cluster, ElastiCache emits metrics at each node level, including both host-level metrics and cache metrics.

For more information on host-level metrics, see Host-Level Metrics.

For more information on node-level metrics, see Metrics for Valkey and Redis OSS.

## Events for node-based clusters (Valkey and Redis OSS)

ElastiCache logs events that relate to your node-based clusters. When working with node-based clusters, you can view your cluster events in the ElastiCache console, using the AWS CLI, or using Amazon Simple Notification Service (SNS). Node-based cluster events are not published to Amazon EventBridge.

Node-based cluster event information includes the date and time of the event, the source name and source type of the event, and a description of the event. You can easily retrieve events from the log using the ElastiCache console, the AWS CLI describe-events command, or the ElastiCache API action DescribeEvents.

Viewing ElastiCache events (Console)

The following procedure displays events using the ElastiCache console.

To view events using the ElastiCache console

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/
2. To see a list of all available events, in the navigation pane, choose Events.
3. On the Events screen each row of the list represents one event and displays the event source, the event type, the GMT time of the event, and a description of the event. Using the Filter you can specify whether you want to see all events, or just events of a specific type in the event list.

Viewing ElastiCache events (AWS CLI)

To generate a list of ElastiCache events using the AWS CLI, use the command describe-events. You can use optional parameters to control the type of events listed, the time frame of the events listed, the maximum number of events to list, and more.

The following code lists up to 40 node-based cluster events.

```
aws elasticache describe-events --source-type cache-cluster --max-items 40
```

The following code lists all events for node-based caches for the past 24 hours (1440 minutes).

```
aws elasticache describe-events --source-type cache-cluster --duration 1440
```

Node-based cluster events

This section contains the list of events you can expect to receive for your node-based clusters.

The following ElastiCache events trigger Amazon SNS notifications. For information on event details, see Viewing ElastiCache events.

| Event Name | Message | Description |
|---|---|---|
| ElastiCache:AddCacheNodeComplete | ElastiCache:AddCacheNodeComplete : *cache-cluster* | A cache node has been added to the cluster and is ready for use. |
| ElastiCache:CacheNodeProvisioningFailedDueToSubnetIPExhaustion | ElastiCache:CacheNodeProvisioningFailedDueToSubnetIPExhaustion *cluster-name* -0001-005 | Amazon ElastiCache could not add a cache node because the subnet does not have enough available IP addresses. |
| ElastiCache:CacheClusterParametersChanged | ElastiCache:CacheClusterParametersChanged : *cluster-name* | One or more cluster parameters have been changed. |
| ElastiCache:CacheClusterProvisioningComplete | ElastiCache:CacheClusterProvisioningComplete *cluster-name-0001-005* | The provisioning of a cluster is completed, and the cache nodes in the cache cluster are ready to use. |
| ElastiCache:CacheClusterProvisioningFailed due to incompatible network state | ElastiCache:CacheClusterProvisioningFailed : *cluster-name* | An attempt was made to launch a new cluster into a nonexistent virtual private cloud (VPC). |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CacheClusterScalingComplete | `CacheClusterScalingComplete : cluster-name` | Scaling for cache-cluster completed successfully. |
| ElastiCache:CacheClusterScalingFailed | ElastiCache:CacheClusterScalingFailed : *cluster-name* | Scale-up operation on cache-cluster failed. |
| ElastiCache:CacheClusterSecurityGroupModified | `ElastiCache:CacheClusterSecurityGroupModified : cluster-name` | One of the following events has occurred:<br>• The list of cache security groups authorized for the cluster has been modified.<br>• One or more new EC2 security groups have been authorized on any of the cache security groups associated with the cluster.<br>• One or more EC2 security groups have been revoked from any of the cache security groups associated with the cluster. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CacheNodeReplaceStarted | `ElastiCache:CacheNodeReplaceStarted : cluster-name` | ElastiCache has detected that the host running a cache node is degraded or unreachable and has started replacing the cache node.<br><br>**Note**<br>The DNS entry for the replaced cache node is not changed.<br><br>In most instances, you do not need to refresh the server-list for your clients when this event occurs. However, some cache client libraries may stop using the cache node even after ElastiCache has replaced the cache node; in this case, the application should refresh the server-list when this event occurs. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CacheNodeReplaceComplete | `ElastiCache:CacheNodeReplaceComplete : cluster-name` | ElastiCache has detected that the host running a cache node is degraded or unreachable and has completed replacing the cache node.<br><br>**Note**<br>The DNS entry for the replaced cache node is not changed.<br><br>In most instances, you do not need to refresh the server-list for your clients when this event occurs. However, some cache client libraries may stop using the cache node even after ElastiCache has replaced the cache node; in this case, the application should refresh the server-list when this event occurs. |
| ElastiCache:CacheNodesRebooted | `ElastiCache:CacheNodesRebooted : cluster-name` | One or more cache nodes has been rebooted.<br><br>Message (Memcached): `"Cache node %s shutdown"` Then a second message: `"Cache node %s restarted"` |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CertificateRenewalComplete (Valkey or Redis OSS only) | `ElastiCache:CertificateRenewalComplete` | The Amazon CA certificate was successfully renewed. |
| ElastiCache:CreateReplicationGroupComplete | `ElastiCache:CreateReplicationGroupComplete : cluster-name` | The replication group was successfully created. |
| ElastiCache:DeleteCacheClusterComplete | `ElastiCache:DeleteCacheClusterComplete : cluster-name` | The deletion of a cluster and all associated cache nodes has completed. |
| ElastiCache:FailoverComplete (Valkey or Redis OSS only) | `ElastiCache:FailoverComplete : mycluster` | Failover over to a replica node was successful. |
| ElastiCache:ReplicationGroupIncreaseReplicaCountFinished | `ElastiCache:ReplicationGroupIncreaseReplicaCountFinished : cluster-name-0001-005` | The number of replicas in the cluster has been increased. |
| ElastiCache:ReplicationGroupIncreaseReplicaCountStarted | `ElastiCache:ReplicationGroupIncreaseReplicaCountStarted : cluster-name-0003-004` | The process of adding replicas to your cluster has begun. |
| ElastiCache:NodeReplacementCanceled | `ElastiCache:NodeReplacementCanceled : cluster-name` | A node in your cluster that was scheduled for replacement is no longer scheduled for replacement. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:NodeReplacementRescheduled | `ElastiCache:NodeReplacementRescheduled : cluster-name` | A node in your cluster previously scheduled for replacement has been rescheduled for replacement during the new window described in the notification.<br><br>For information on what actions you can take, see Replacing nodes (Valkey and Redis OSS). |
| ElastiCache:NodeReplacementScheduled | `ElastiCache:NodeReplacementScheduled : cluster-name` | A node in your cluster is scheduled for replacement during the window described in the notification.<br><br>For information on what actions you can take, see Replacing nodes (Valkey and Redis OSS). |
| ElastiCache:RemoveCacheNodeComplete | `ElastiCache:RemoveCacheNodeComplete : cluster-name` | A cache node has been removed from the cluster. |
| ElastiCache:ReplicationGroupScalingComplete | `ElastiCache:ReplicationGroupScalingComplete : cluster-name` | Scale-up operation on replication group completed successfully. |
| ElastiCache:ReplicationGroupScalingFailed | `"Failed applying modification to cache node type to %s."` | Scale-up operation on replication group failed. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:ServiceUpdateAvailableForNode | `"Service update is available for cache node %s."` | A self-service update is available for the node. |
| ElastiCache:SnapshotComplete (Valkey or Redis OSS only) | `ElastiCache:SnapshotComplete : cluster-name` | A cache snapshot has completed successfully. |
| ElastiCache:SnapshotFailed (Valkey or Redis OSS only) | `SnapshotFailed : cluster-name` | A cache snapshot has failed. See the cluster's cache events for more a detailed cause.<br><br>If you describe the snapshot, see `DescribeSnapshots` , the status will be `failed`. |

# Monitoring ElastiCache with OpenTelemetry metrics

Amazon ElastiCache emits OpenTelemetry (OTLP) metrics for node-based Valkey replication groups to Amazon CloudWatch. These metrics give you a detailed view of your replication group, with a broad set of metrics from the cache engine and the host, rich attributes for filtering and aggregation, and support for querying with PromQL.

A core set of vended metrics is emitted automatically, at 60-second granularity, at no additional cost. You can also select any metric with a resource metrics configuration to be emitted at 15-second granularity. Selected metrics are charged by CloudWatch. For more information, see Amazon CloudWatch pricing.

## How OpenTelemetry metrics work

Each node in your replication group collects telemetry from the cache engine and from the host that the node runs on, and emits it to CloudWatch as OpenTelemetry metrics. Metrics are emitted per node, and every datapoint carries attributes that identify the account, Region, Availability Zone, replication group, shard, and node that produced it.

Vended metrics are emitted automatically, at 60-second granularity, as soon as the replication group is running. You do not need to configure anything, and they are not charged. Querying vended metrics through the CloudWatch API, from Grafana, or from alarms incurs query charges. For more information, see Querying your metrics.

Detailed metrics are emitted only while you select them. To begin emitting them, create a resource metrics configuration for the replication group and list the metrics you want. Those metrics are then emitted at 15-second granularity and charged by CloudWatch. Selecting a metric that is already vended raises its granularity from 60 seconds to 15 seconds, and it is charged while it remains selected. For more information, see Turning on detailed monitoring.

To see which metrics are emitted automatically, see OpenTelemetry metrics reference for Amazon ElastiCache.

After the metrics reach CloudWatch, you work with them the same way as any other OpenTelemetry metrics in CloudWatch. You can view them in the prebuilt ElastiCache Insights dashboard, query them with PromQL, explore them in Query Studio, create alarms, and build your own dashboards. For information about the dashboard, see Viewing metrics in the ElastiCache Insights dashboard. For information about querying, see Querying your metrics.

## Metric attributes

Every datapoint carries attributes that let you filter and aggregate your metrics at query time. Attributes fall into the following categories.

- **Resource attributes** identify the AWS account, Region, and Availability Zone, and the replication group, shard, and node that produced the datapoint.
- **Instrumentation scope attributes** identify ElastiCache as the source of the metric.
- **Datapoint attributes** provide per-metric dimensions. Every metric reports the replication role of the node that emitted it, and many metrics add attributes that break the metric down further, such as a command name or a network direction.

For the complete list of attributes and the values they take, see OpenTelemetry metrics reference for Amazon ElastiCache.

## Supported configurations

All node-based Valkey replication groups can emit OpenTelemetry metrics, on any Valkey version, with the exception of replication groups on AWS Outposts. Serverless caches, Redis OSS, and Memcached do not emit OpenTelemetry metrics.

In a global datastore, select metrics separately for each regional replication group.

## Turning on detailed monitoring

Turn on detailed monitoring when you need a metric that is not emitted automatically, or a granularity of 15 seconds.

To turn on detailed monitoring, create a resource metrics configuration for the replication group, identified by its ARN. A replication group can have only one configuration at a time. For more information about resource metrics configurations, see CloudWatch detailed monitoring for OpenTelemetry metrics in the *Amazon CloudWatch User Guide*.

> **Note**
>
> Creating, retrieving, updating, and deleting a configuration requires the corresponding `cloudwatch:*ResourceMetricsConfiguration` permissions. Because these operations do not support resource-level permissions, an IAM policy must use `"Resource": "*"`. Scope access with the `cloudwatch:ResourceArn` condition key

instead. These are Amazon CloudWatch API operations, not ElastiCache API operations. For the required permissions and for more information, see CloudWatch detailed monitoring for OpenTelemetry metrics in the *Amazon CloudWatch User Guide*.

## Using the console

You can turn on detailed monitoring for an existing replication group from its **Metrics** tab, or while you create a replication group.

**To turn on detailed monitoring for an existing replication group**

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/, and choose your replication group.
2. On the **Metrics** tab, choose **Detailed**.
3. Choose **Configure detailed metrics**.
4. Select the metrics that you want the replication group to emit, then save your changes.

Until you select metrics, the **Detailed** view shows **No detailed metrics configured**.

While you create a replication group, choose **Detailed monitoring** in the **Monitoring** section, then select your metrics.

## Using the AWS CLI

To emit specific metrics, create a configuration with the `--metric-selections` parameter and list the metrics by name.

```
aws cloudwatch create-resource-metrics-configuration \
    --resource-arn arn:aws:elasticache:us-east-1:111122223333:replicationgroup:my-
cluster \
    --metric-selections '[{"IncludeMetrics": ["valkey.memory.used",
 "valkey.commands.processed"]}]'
```

If you omit `--metric-selections`, all available metrics are selected. This includes every vended metric, which moves each one to 15-second granularity and starts charging for it. Because charges are based on the volume of datapoints emitted, we recommend that you select only the metrics that you need.

```
aws cloudwatch create-resource-metrics-configuration \
    --resource-arn arn:aws:elasticache:us-east-1:111122223333:replicationgroup:my-
cluster
```

To see which metrics a replication group is currently emitting, retrieve its configuration. If no configuration exists, the command returns an error.

```
aws cloudwatch get-resource-metrics-configuration \
    --resource-arn arn:aws:elasticache:us-east-1:111122223333:replicationgroup:my-
cluster
```

To change which metrics are selected, update the configuration.

```
aws cloudwatch update-resource-metrics-configuration \
    --resource-arn arn:aws:elasticache:us-east-1:111122223333:replicationgroup:my-
cluster \
    --metric-selections '[{"IncludeMetrics": ["valkey.memory.used",
 "valkey.keyspace.hits"]}]'
```

**Important**

The `--metric-selections` parameter completely replaces the previous value. To add a metric to an existing selection, list every metric that you want to emit, not only the new one.

To stop emitting metrics, delete the configuration.

```
aws cloudwatch delete-resource-metrics-configuration \
    --resource-arn arn:aws:elasticache:us-east-1:111122223333:replicationgroup:my-
cluster
```

You can also create a configuration with the `AWS::CloudWatch::ResourceMetricsConfiguration` CloudFormation resource type. For more information, see AWS::CloudWatch::ResourceMetricsConfiguration in the *CloudFormation Template Reference Guide*.

## After you turn on detailed monitoring

Changes to a resource metrics configuration take a few minutes to take effect.

When you delete a replication group, its resource metrics configuration is deleted automatically.

> **Note**
>
> Charges are based on the volume of datapoints emitted for the metrics you select, not on the number of metrics you select. High-cardinality metrics that are broken down by an attribute, such as `valkey.command.calls`, emit a datapoint for each attribute value, so their cost depends on your workload. Metrics that you have not selected are not charged.

## Viewing metrics in the ElastiCache Insights dashboard

ElastiCache Insights is a prebuilt dashboard in the CloudWatch console that visualizes your OpenTelemetry metrics. You can use the dashboard as is. You can also copy the dashboard and modify it.

**To open the ElastiCache Insights dashboard**

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, under **Infrastructure Monitoring**, choose **ElastiCache Insights**.

To open the dashboard directly, go to https://console.aws.amazon.com/cloudwatch/home#dashboards/insights/elasticache.

You can also open the dashboard from the **Metrics** tab of a replication group in the ElastiCache console.

Saving a copy as a custom dashboard lets you add or remove widgets, change the metrics and attributes that a widget displays, or combine ElastiCache metrics with metrics from other sources. Changes that you make to a copy do not affect the prebuilt dashboard. For more information, see Using Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

Using the ElastiCache Insights dashboard is free. If you save a copy, the copy is billed as a CloudWatch custom dashboard. For more information, see Amazon CloudWatch pricing.

## Querying your metrics

You query ElastiCache OpenTelemetry metrics using PromQL, which supports query-time aggregation and calculation across your replication groups, shards, and nodes. You can run queries

interactively in Query Studio, use them to create CloudWatch alarms, and add them to dashboards. You can also query them from Amazon Managed Grafana by adding a Prometheus data source that points to the CloudWatch PromQL endpoint.

For more information, see the following topics in the *Amazon CloudWatch User Guide*.

- Query metrics with PromQL
- PromQL querying

For information about querying from Amazon Managed Grafana, see Query Amazon CloudWatch metrics using PromQL in the *Amazon Managed Grafana User Guide*.

> **Note**
>
> Vended metrics are free to emit, but querying any metric, vended or detailed, can incur charges. PromQL queries executed through the AWS console (including CloudWatch dashboards and Query Studio) are free of charge. Queries executed through the CloudWatch API, including from Grafana, incur query charges based on the number of samples scanned. CloudWatch alarms that evaluate PromQL expressions are charged for the alarm and for the samples scanned on each evaluation. For more information, see Amazon CloudWatch pricing.

> **Note**
>
> Use a time range that contains multiple datapoints. A shorter range can return noisy data, or no data at all. We recommend a time range of at least `[5m]` for metrics emitted every 60 seconds, or at least `[1m]` for metrics emitted every 15 seconds.

## Key PromQL capabilities

The following example queries demonstrate key operations that PromQL enables for observability. Replace `my-cluster` with your replication group ID and `my-cluster-0001-001` with a node ID.

For a larger set of ready-to-copy queries organized by topic, see Example queries for common monitoring tasks. For the full list of functions and operators that CloudWatch supports, see Query metrics with PromQL in the *Amazon CloudWatch User Guide*.

**Select a time series: Memory used by a node**

```
{"valkey.memory.used",
 "@resource.aws.elasticache.replication_group.id"="my-cluster",
 "@resource.aws.elasticache.node.id"="my-cluster-0001-001"}
```

A query is a metric name followed by any number of attribute matchers, enclosed in braces. Names are quoted because they contain dots, and resource attributes take the `@resource.` prefix. Omitting the node matcher returns one time series per node.

**Basic operations: Memory used by a node as a percentage of its limit**

```
100 * {"valkey.memory.used",
       "@resource.aws.elasticache.replication_group.id"="my-cluster",
       "@resource.aws.elasticache.node.id"="my-cluster-0001-001"}
    / {"valkey.memory.max",
       "@resource.aws.elasticache.replication_group.id"="my-cluster",
       "@resource.aws.elasticache.node.id"="my-cluster-0001-001"}
```

PromQL pairs up time series that have identical attributes and calculates a result for each pair. Both metrics here are emitted by the same node, so they pair up; if the two sides carried different attributes, the query would return no result.

**Convert a counter to a rate: Inbound network throughput of each node**

```
rate({"system.network.io",
      "network.io.direction"="receive",
      "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

Because the query matches only on the replication group, it returns one series for each node, and all of them appear on the same chart. Counters are cumulative, so `rate` converts one into a per-second rate over the time range in brackets. It also accounts for the total resetting to zero when a node restarts or fails over.

**Aggregations**

Aggregation operators combine several time series into fewer series, either collapsing them into a single result or grouping them into buckets.

**Aggregate all matching time series: Total memory used across a replication group**

```
sum({"valkey.memory.used",
     "valkey.role"="primary",
     "@resource.aws.elasticache.replication_group.id"="my-cluster"})
```

Filtering on `valkey.role` counts only primaries, which hold the data that every replica in their shard replicates. `valkey.role` is a datapoint attribute, so unlike the resource attributes it takes no prefix.

**Aggregate into buckets: Transactions per second (TPS) of each shard**

```
sum by ("@resource.aws.elasticache.shard.id") (
  rate({"valkey.commands.processed",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

A `by` clause aggregates into groups rather than into a single value, so this query returns one TPS figure for each shard, combining every node in that shard. A `by` clause also keeps only the attributes it names and discards the rest, so these results no longer identify the Region or account. Use a `without` clause to keep every attribute except the ones you name.

**Aggregate all but one attribute: CPU utilization of the cache engine on each node**

```
100 * sum without ("cpu.mode") (
        rate({"process.cpu.time",
              "thread.type"="main",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

`rate` converts cumulative CPU seconds into CPU seconds per second, and summing `without` the `cpu.mode` attribute combines user and system time, giving the percentage of a single vCPU consumed by the engine's main thread. Using `without` rather than `by` keeps the resource attributes, so each node reports separately. Only the main thread is reported, so this query does not include CPU time consumed by I/O threads.

**Combine functions, aggregations and operations: Average latency of each command**

```
1e6 * sum by ("valkey.command") (
        rate({"valkey.command.duration",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / (sum by ("valkey.command") (
```

```
        rate({"valkey.command.calls",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])) > 0)
```

ElastiCache emits the cumulative execution time and the cumulative call count for each command, so dividing one rate by the other gives the average duration of a call, in microseconds. Both sides are aggregated by `valkey.command` so that each command's execution time is divided by its own call count, and the `> 0` comparison excludes commands that were not called during the time range.

**Match attributes with a regular expression: TPS for a set of commands across replication groups**

```
sum by ("valkey.command") (
  rate({"valkey.command.calls",
        "valkey.command"=~"GET|MGET|HGET|LRANGE",
        "@resource.aws.elasticache.replication_group.id"=~"prod-.*"}[5m]))
```

The =~ operator matches an attribute against a regular expression, and works on any attribute; use `!~` to exclude values that match. Expressions are anchored, so `"valkey.command"=~"GET"` matches only `GET` and not `GETRANGE` — end an expression with `.*` to match a prefix.

**Compare across nodes: Each node's share of its shard's traffic**

```
  rate({"valkey.commands.processed",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
/ on ("@resource.aws.elasticache.shard.id") group_left
  sum by ("@resource.aws.elasticache.shard.id") (
    rate({"valkey.commands.processed",
          "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

Dividing each node's command rate by the total rate of its shard gives the share of the shard's traffic that each node handles. Aggregating by shard reduces the denominator to one value per shard, and on restricts the match to the shard attribute so that each node is divided by the total of its own shard. `group_left` allows that single shard total to be matched against every node in the shard.

**Other useful functions**

**Read a percentile from a histogram: Peak inbound network throughput**

```
histogram_quantile(0.99,
```

```
  {"system.network.io.rate",
   "network.io.direction"="receive",
   "@resource.aws.elasticache.replication_group.id"="my-cluster"})
```

Two metrics, `system.network.io.rate` and `system.network.packet.rate`, are exponential histograms rather than single values. The network interface is sampled once a second, and each datapoint carries the distribution of those per-second rates, so `histogram_quantile` reads a percentile from it. The first argument is the quantile, where `0.99` is the 99th percentile. Reading a high percentile reveals one-second bursts that an averaged value would smooth away.

**Smooth a series, and find a peak: Fragmentation over a window, and peak memory over a day**

```
avg_over_time(
  {"valkey.memory.allocator_frag_ratio",
   "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

`avg_over_time` averages a metric across a time window, which suppresses brief spikes that would otherwise dominate a chart or trigger an alarm. `max_over_time` is its counterpart and reports the highest value in the window, which is how you find a peak over a period that you choose:

```
max_over_time(
  {"valkey.memory.used",
   "@resource.aws.elasticache.replication_group.id"="my-cluster"}[24h])
```

Both functions require a range in square brackets, and the range is limited to 7 days.

**Find highest values: The busiest nodes across every replication group**

```
topk(5,
  sum without ("cpu.mode") (
    rate({"process.cpu.time", "thread.type"="main"}[5m])))
```

`topk` returns only the highest N series, which turns a query that would return hundreds of series into a short, readable list. Because this query has no replication group matcher, it ranks nodes across every replication group that emits the metric. `bottomk` does the same for the lowest values.

> **Note**
>
> A query can return at most 500 time series. On a large replication group, a query that breaks results down by both node and command can exceed that limit and return a truncated response. Aggregate the result, or narrow it with a more specific label matcher.

## Creating alarms from queries

You can create a CloudWatch alarm from a PromQL query, which lets you alarm on values that are calculated at query time, such as a percentage of a limit or a comparison between nodes.

Three characteristics of a PromQL alarm affect how you write the query.

**The threshold is part of the query.** You add a comparison operator and a value to the query, and the query returns only the series that satisfy the comparison. For example, this query returns a series only for nodes that are using more than 90 percent of their memory limit:

```
100 * {"valkey.memory.used",    "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
    / {"valkey.memory.max",     "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
    > 90
```

**One alarm can watch every node.** Each series that the query returns is tracked independently as a contributor, and the alarm enters the ALARM state when any contributor is breaching. You do not need a separate alarm for each node, and you do not need to aggregate the query to a single value. The attributes of a breaching series identify which node or shard is responsible.

**You control sensitivity with durations.** A PromQL alarm takes an evaluation interval, a pending period, and a recovery period:

| Setting | Effect |
|---|---|
| Evaluation interval | How often the query runs. Valid values are 10, 20, 30, or any multiple of 60, up to 3600 seconds |
| Pending period | How long a contributor must breach continuously before the alarm enters the ALARM state |

| Setting | Effect |
| --- | --- |
| Recovery period | How long a contributor must stop breaching before it returns to the OK state |

The following command creates an alarm from the memory query above.

```
aws cloudwatch put-metric-alarm \
    --alarm-name my-cluster-memory-high \
    --evaluation-criteria '{"PromQLCriteria":
{"Query":"<query>","PendingPeriod":300,"RecoveryPeriod":600}}' \
    --evaluation-interval 60
```

> **Note**
>
> Set the evaluation interval no shorter than the interval at which the metric is emitted. A shorter interval re-evaluates the same datapoints. For a vended metric, use 60 seconds or longer. The range in a query, such as `[5m]`, sets how much data each evaluation covers, not how often the alarm evaluates, so an alarm can evaluate more often than its range.

Use the pending period, rather than a long averaging window inside the query, to keep brief spikes from raising an alarm. A pending period of 300 seconds requires 5 minutes of continuous breaching, which is easier to reason about than smoothing the query.

Because a contributor is considered recovered once its series stops being returned, an alarm also recovers when the series disappears for any other reason. A metric that is broken down by an attribute only produces a series while your workload generates that attribute value, so an alarm on a specific command or error type returns to OK when your workload stops using it.

## Learning more about PromQL

CloudWatch PromQL supports many more functions and features than these examples show, including additional aggregation operators, comparisons against earlier time periods, and functions for smoothing and forecasting. For the full language reference and further examples, see Query metrics with PromQL and PromQL querying in the *Amazon CloudWatch User Guide*.

## Pricing

Vended metrics are emitted at no additional cost. Metrics that you select for detailed monitoring are billed by Amazon CloudWatch by the volume of datapoints emitted. For more information, see Amazon CloudWatch pricing.

Querying metrics outside the console and evaluating alarms are charged separately. For more information, see Querying your metrics.

# OpenTelemetry metrics reference for Amazon ElastiCache

This topic lists the OpenTelemetry metrics that Amazon ElastiCache emits for node-based Valkey replication groups, along with the attributes that accompany them. For an introduction to the feature, including how to enable it, see Monitoring ElastiCache with OpenTelemetry metrics.

Most metrics are emitted by every node in the replication group and are identified by a metric name, a tier, a type, a unit, and a set of attributes. Some metrics are emitted only under specific conditions, as noted in the metric's description.

## Vended and detailed metrics

The **Tier** column of the tables in this topic shows how each metric becomes available.

| Tier | Description |
| --- | --- |
| Vended | Emitted automatically, at no additional cost. |
| Detailed only | Emitted only while the metric is selected in a resource metrics configuration. |

Vended metrics cover the measurements most replication groups need, and they are available without any action on your part. Turn on detailed monitoring when you want a finer granularity, or a metric that the vended tier does not include. For more information, see Turning on detailed monitoring.

## Common attributes

Every metric carries a common set of attributes that identify where the datapoint came from. Use these attributes to filter and aggregate your metrics in PromQL.

When CloudWatch ingests OpenTelemetry metrics, the OTLP data model is flattened into PromQL labels. The label name you use in a query depends on the OTLP scope that the attribute belongs to.

| OTLP scope | Prefix for attributes | Example |
|---|---|---|
| Resource | `@resource.` | `@resource.aws.elasticache.node.id` |
| Instrumentation scope | `@instrumentation.` | `@instrumentation.@name` |
| Datapoint | None, or `@datapoint.` | `valkey.role` |

For more information about the label structure and query syntax, see PromQL querying in the *Amazon CloudWatch User Guide*.

**Resource attributes**

Resource attributes identify the account, the location, and the ElastiCache resource that produced the datapoint. In a PromQL query, prefix these attributes with `@resource.`.

| Attribute | Example value | Description |
|---|---|---|
| `cloud.provider` | `aws` | The cloud provider. |
| `cloud.region` | `us-east-1` | The AWS Region that the node runs in. |
| `cloud.availability_zone_id` | `use1-az1` | The ID of the Availability Zone that the node runs in. |
| `cloud.account.id` | `111122223333` | The AWS account ID that owns the replication group. |
| `cloud.resource_id` | `arn:aws:elasticach` | The ARN of the replication group. |

| Attribute | Example value | Description |
|---|---|---|
| | e:us-east-1:111122223333:replicationgroup:my-cluster | |
| aws.elasticache.replication_group.id | my-cluster | The ID of the replication group. |
| aws.elasticache.shard.id | my-cluster-0001 | The ID of the shard that contains the node. |
| aws.elasticache.node.id | my-cluster-0001-001 | The ID of the node that emitted the datapoint. |
| aws.elasticache.engine.name | valkey | The name of the cache engine. |
| aws.elasticache.engine.version | 8.1.0 | The version of the cache engine. |

### Instrumentation scope attributes

The instrumentation scope identifies ElastiCache as the source of the metrics. The scope name and version are scope *fields*, which you query using the double @ form.

| PromQL label | Example value | Description |
|---|---|---|
| @instrumentation.@name | cloudwatch.aws/elasticache | The instrumentation scope name. |

| PromQL label | Example value | Description |
|---|---|---|
| `@instrumentation.@version` | `1.0.0` | The instrumentation scope version. |
| `@instrumentation.cloudwatch.solution` | `vended` \| `detailed` | The tier that emitted the datapoint. Use it to identify which datapoints were charged. |

#### Datapoint attributes

Datapoint attributes are queried as bare labels, or optionally with the `@datapoint.` prefix. The following attribute is present on every metric.

| Attribute | Values | Description |
|---|---|---|
| `valkey.role` | `primary` \| `replica` | The replication role of the node that emitted the datapoint. |

Many metrics carry additional datapoint attributes that break the metric down further. These are listed in the **Additional attributes** column of the metric tables.

### Metric types

The **Type** column of the metric tables gives the OpenTelemetry instrument that each metric is emitted as, which determines how you query it.

| Type | Meaning | Querying |
|---|---|---|
| `Counter` | A cumulative sum that only increases, accumulated over the lifetime of the node, | Use `rate` for a per-second value, or `increase` for a total over a time range. The cumulative value itself is rarely useful. Values are additive, so aggregating across nodes or shards with `sum` is meaningful |

| Type | Meaning | Querying |
| --- | --- | --- |
| | such as the number of commands processed | |
| `UpDownCounter` | A sum that can increase or decrease, such as the number of connected clients | Query it directly. Values are additive, so aggregating across nodes with `sum` is meaningful |
| `Gauge` | A value sampled at a point in time, such as a ratio or a percentage | Query it directly. Values are not additive, so aggregating across nodes with `sum` is usually meaningless. Use `avg`, `max` or `min` instead |
| `ExponentialHistogram` | A distribution of values rather than a single value | Use `histogram_quantile` to read a percentile from it |

## Metrics

The `valkey.*` metrics are reported by the cache engine through the `INFO` command, and the **INFO field** column gives the field that each metric is collected from. For information about these fields, see INFO in the Valkey command reference. The `system.*` and `process.*` metrics are reported by the operating system: `system.*` metrics describe the host as a whole, and `process.*` metrics describe the Valkey process specifically.

The ElastiCache console shows a display name for each metric alongside the metric name that appears here.

### Clients

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.clients.connected | Vended | UpDownCounter | {client} | connected_clients | — | The number of current client connections, excluding replicas. |
| valkey.clients.max | Vended | Gauge | {client} | maxclients | — | The configured maxclients limit. This limit applies to the sum of client connections, replica connections, and cluster bus connections. |
| valkey.clients.blocked | Vended | UpDownCounter | {client} | blocked_clients | — | The number of clients pending on a blocking call, such as BLPOP, BLMOVE, or BZPOPMIN. |
| valkey.clients.tracking | Detailed only | UpDownCounter | {client} | tracking_clients | — | The number of clients in client-side caching tracking mode. |
| valkey.clients.pubsub | Detailed only | UpDownCounter | {client} | pubsub_clients | — | The number of clients in publish/subscribe mode. |
| valkey.clients.watching | Detailed only | UpDownCounter | {client} | watching_clients | — | The number of clients with an active WATCH. |
| valkey.clients.in_ | Detailed only | UpDownCounter | {client} | clients_in_timeout_table | — | The number of clients tracked in the timeout table. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| timeout_table | | | | | | |
| valkey.clients.cluster_connections | Detailed only | UpDownCounter | {connection} | cluster_connections | — | An approximation of the number of sockets used by the cluster bus. |
| valkey.clients.max_input_buffer | Detailed only | Gauge | By | client_recent_max_input_buffer | — | The size of the largest recent client input buffer. |
| valkey.clients.max_output_buffer | Detailed only | Gauge | By | client_recent_max_output_buffer | — | The size of the largest recent client output buffer. |
| valkey.clients.evicted | Detailed only | Counter | {event} | evicted_clients | — | The number of clients evicted because of the maxmemory-clients limit. |
| valkey.clients.output_buffer_disconnections | Detailed only | Counter | {event} | client_output_buffer_limit_disconnections | — | The number of clients disconnected for exceeding the output buffer limit. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.clients.query_buffer_disconnections | Detailed only | Counter | {event} | client_query_buf_limit_disconnections | — | The number of clients disconnected for exceeding the query buffer limit. |
| valkey.clients.reply_buffer_expands | Detailed only | Counter | {event} | reply_buffer_expands | — | The number of client reply buffer expansion events. |
| valkey.clients.reply_buffer_shrinks | Detailed only | Counter | {event} | reply_buffer_shrinks | — | The number of client reply buffer shrink events. |

### Connections

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.connections.received | Vended | Counter | {connection} | total_connections_received | — | The total number of connections accepted by the node. |
| valkey.connections.rejected | Vended | Counter | {connection} | rejected_connections | — | The number of connections rejected because the `maxclients` limit was reached. |

### Commands

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.commands.processed | Vended | Counter | {commar | total_commands_processed | — | The total number of commands processed by the node. |

### Commands by command name

The following metrics break command activity down by individual command. They are collected from the per-command cmdstat_<command> entries that INFO returns, where <command> is the name of the command.

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.command.calls | Vended | Counter | {call} | cmdstat_command calls | valkey.command, valkey.command.type, valkey.command.category | The number of calls for each command. |
| valkey.command.duration | Vended | Counter | s | cmdstat_command usec | valkey.command, valkey.command.type, valkey. | The total execution time for each command. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| | | | | | mmand.c<br>egory | |
| valkey.command.rejected_calls | Detailed only | Counter | {call} | cmdstat<br>command<br>rejecte<br>calls | valkey.<br>mmand ,<br>valkey.<br>mmand.t<br>e ,<br>valkey.<br>mmand.c<br>egory | The number of rejected calls for each command. |
| valkey.command.failed_calls | Detailed only | Counter | {call} | cmdstat<br>command<br>failed_<br>lls | valkey.<br>mmand ,<br>valkey.<br>mmand.t<br>e ,<br>valkey.<br>mmand.c<br>egory | The number of failed calls for each command. |

The `valkey.command` attribute contains the command name in uppercase, for example `SET` or `GET`. For a command that has subcommands, the attribute contains the command name and the subcommand name separated by a space, for example `OBJECT ENCODING` or `CLUSTER INFO`.

The `valkey.command.type` and `valkey.command.category` attributes classify each command, so that you can aggregate command activity without listing command names. Use `valkey.command.type` to separate reads from writes, and `valkey.command.category` to group commands by the data type or the functional area that they belong to.

| Attribute | Values | Description |
| --- | --- | --- |
| valkey.command.type | read \| write | Whether the command reads or writes data. |
| valkey.command.category | string, hash, list, set, sortedset, bitmap, hyperloglog, geo, stream, json, bloomfilter, search, generic, pubsub, scripting, transactions, connection, cluster, server | The data type or the functional area that the command belongs to. |

**Note**

Not every command carries both attributes. valkey.command.type is present only for commands that read or write data, so commands such as PING, SUBSCRIBE, and MULTI are emitted without it. Write queries so that they do not depend on an attribute that a command does not have. For example, a query that matches valkey.command.type returns only the commands that report a type. A given valkey.command.category value appears only while your workload calls commands in that category.

### Keyspace

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.keyspace.hits | Vended | Counter | {hit} | keyspace_hits | — | The number of successful key lookups. |
| valkey.keyspace.misses | Vended | Counter | {miss} | keyspace_misses | — | The number of failed key lookups. |

### Keys

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.keys.expired | Vended | Counter | {key} | expired_keys | — | The total number of keys that have expired. |
| valkey.keys.evicted | Vended | Counter | {key} | evicted_keys | — | The number of keys evicted because of the maxmemory limit. |
| valkey.keys.tracked | Vended | UpDownCounter | {key} | tracking_total_keys | — | The number of keys tracked for client-side caching. |
| valkey.keys.expired_fields | Vended | Counter | {field} | expired_fields | — | The number of hash fields that expired through a per-field time to live (TTL). |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.keys.expired_stale_percentage | Detailed only | Gauge | % | expired_stale_pe | — | The estimated percentage of keys that have probably expired but are still in the keyspace. |
| valkey.keys.watched | Detailed only | UpDownCounter | {key} | total_watched_ke | — | The number of keys under WATCH across all clients. |
| valkey.keys.blocking | Detailed only | UpDownCounter | {key} | total_blocking_keys | — | The number of distinct keys that have a blocked client. |
| valkey.keys.blocking_on_nokey | Detailed only | UpDownCounter | {key} | total_blocking_keys_on_nokey | — | The number of blocking keys that have one or more clients waiting to be unblocked when the key is deleted. |

**Memory**

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.memory.used | Vended | UpDownCounter | By | used_memory | — | The total memory used by the cache engine. |
| valkey.memory.used.dataset | Vended | UpDownCounter | By | used_memory_dataset | — | The memory used by the dataset, which is used memory minus overhead. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.memory.max | Vended | Gauge | By | maxmemo | — | The configured maxmemory limit. |
| valkey.memory.fragmentation_ratio | Vended | Gauge | 1 | mem_fra entatio ratio | — | The ratio of resident set size to used memory. |
| valkey.memory.allocator_frag_bytes | Vended | UpDownCounter | By | allocat _frag_b es | — | The allocator-level fragmentation, in bytes. |
| valkey.memory.allocator_frag_ratio | Vended | Gauge | 1 | allocat _frag_r io | — | The allocator fragmentation ratio. This is the true external fragmentation metric, rather than valkey.memory.fragmentation_ratio . |
| valkey.memory.not_counted_for_evict | Vended | UpDownCounter | By | mem_not ounted_ r_evict | — | The used memory that is not counted for key eviction, which is mainly transient replica and append-only file (AOF) buffers. |
| valkey.memory.used.overhead | Detailed only | UpDownCounter | By | used_me ry_over ad | — | The memory used for internal structures and buffers, excluding the dataset. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.memory.used.startup | Detailed only | UpDownCounter | By | used_memory_startup | — | The memory consumed at engine startup. |
| valkey.memory.used.vm_eval | Detailed only | UpDownCounter | By | used_memory_vm_eval | — | The memory used by the script VM engine for the EVAL framework. This memory is not part of used memory. |
| valkey.memory.used.scripts | Detailed only | UpDownCounter | By | used_memory_scripts | — | The memory overhead of EVAL scripts and functions. This memory is part of used memory. |
| valkey.memory.used.functions | Detailed only | UpDownCounter | By | used_memory_functions | — | The memory overhead of function scripts. This memory is part of used memory. |
| valkey.memory.fragmentation_bytes | Detailed only | UpDownCounter | By | mem_fragmentation_bytes | — | The fragmentation between resident set size and used memory, in bytes. |
| valkey.memory.clients.normal | Detailed only | UpDownCounter | By | mem_clients_norm | — | The buffer memory used by normal clients. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.memory.clients.replica | Detailed only | UpDownCounter | By | mem_clients_slave | — | The buffer memory used by replica clients. Replica buffers share memory with the replication backlog, so this metric can report 0 when replicas do not increase memory usage. |
| valkey.memory.cluster_links | Detailed only | UpDownCounter | By | mem_cluster_link | — | The buffer memory used by cluster bus links. |
| valkey.memory.replication_backlog | Detailed only | UpDownCounter | By | mem_replication_backlog | — | The memory used by the replication backlog. |
| valkey.memory.replication_buffers | Detailed only | UpDownCounter | By | mem_tot_replication_buffers | — | The total memory used by replication buffers. |
| valkey.memory.lazyfree_pending_objects | Detailed only | UpDownCounter | {object} | lazyfree_pending_objects | — | The number of objects waiting to be freed as a result of UNLINK, or of FLUSHDB and FLUSHALL with the ASYNC option. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.memory.lazyfreed_objects | Detailed only | Counter | {object | lazyfre_object | — | The total number of objects freed lazily. |

### Defragmentation

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.defrag.hits | Vended | Counter | {hit} | active_frag_hi | — | The number of reallocations performed by active defragmentation. |
| valkey.defrag.misses | Detailed only | Counter | {miss} | active_frag_mi es | — | The number of value reallocations that active defragmentation started and then aborted. |
| valkey.defrag.running | Detailed only | Gauge | % | active_frag_ru ing | — | The percentage of CPU that active defragmentation intends to use. The value is 0 when active defragmentation is not running. |
| valkey.defrag.key_hits | Detailed only | Counter | {key} | active_frag_ke hits | — | The number of keys that were actively defragmented. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.defrag.key_misses | Detailed only | Counter | {key} | active_frag_ke misses | — | The number of keys that were skipped by active defragmentation. |
| valkey.defrag.duration | Detailed only | Counter | s | total_a ive_def g_time | — | The total time that memory fragmentation was over the limit. |

### Network

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.network.input | Detailed only | Counter | By | total_r _input_ tes | — | The total bytes read from the network by the cache engine. |
| valkey.network.output | Detailed only | Counter | By | total_r _output ytes | — | The total bytes written to the network by the cache engine. |
| valkey.network.input.replication | Detailed only | Counter | By | total_r _repl_i ut_byte | — | The total replication bytes read. |
| valkey.network.out | Detailed only | Counter | By | total_r _repl_o put_byt | — | The total replication bytes written. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| put.replication | | | | | | |

**Replication**

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.replication.offset | Vended | Gauge | By | master_pl_offs | — | The replication offset of the node. |
| valkey.replication.is_primary | Vended | Gauge | {state} | role | — | Set to 1 if the node is a primary, otherwise 0. |
| valkey.replication.link_healthy | Vended | Gauge | {state} | master_nk_stat | — | Set to 1 if the link from the replica to its primary is up, otherwise 0. Emitted on replicas. |
| valkey.replication.connected_replicas | Detailed only | UpDownnter | {replic | connect_slaves | — | The number of connected replicas. |
| valkey.replication.last_io_ | Detailed only | Gauge | s | master_st_io_sonds_ag | — | The time since the last interaction with the |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| seconds_ago | | | | | | primary. Emitted on replicas. |
| valkey.replication.sync.in_progress | Detailed only | Gauge | {state} | master_nc_in_progress | — | Set to 1 while an initial full synchronization is running, otherwise 0. Emitted on replicas. |
| valkey.replication.sync.left_bytes | Detailed only | Gauge | By | master_nc_leftytes | — | The bytes remaining before an in-progress full synchronization is complete. The value can be negative when the total size is unknown. Emitted on replicas. |
| valkey.replication.backlog_active | Detailed only | Gauge | {state} | repl_balog_acte | — | Set to 1 if the replication backlog is active, otherwise 0. |
| valkey.replication.backlog_size | Detailed only | Gauge | By | repl_balog_siz | — | The configured size of the replication backlog. |
| valkey.replication.backlog_histlen | Detailed only | UpDownnter | By | repl_balog_hisen | — | The number of bytes currently held in the replication backlog. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.replication.replicas_buffer_size | Detailed only | UpDownCounter | By | replica_repl_buffer_size | — | The replication stream data currently accumulated during dual-channel replication. |
| valkey.replication.replicas_waiting_psync | Detailed only | UpDownCounter | {replic | replica_waiting_sync | — | The number of replicas waiting for partial resynchronization during dual-channel replication. |
| valkey.replication.sync.full | Detailed only | Counter | {sync} | sync_fu | — | The number of full resynchronizations served to replicas. |
| valkey.replication.sync.partial_ok | Detailed only | Counter | {sync} | sync_partial_ok | — | The number of partial resynchronizations served successfully. |
| valkey.replication.sync.partial_err | Detailed only | Counter | {sync} | sync_partial_err | — | The number of partial resynchronizations that could not be served. |

### Persistence

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.persistence.rdb.save_in_progress | Vended | Gauge | {state} | rdb_bgsave_in_progress | — | Set to 1 while an RDB save is running, including an RDB save for diskless replication, otherwise 0. |
| valkey.persistence.rdb.changes_since_save | Detailed only | UpDownCounter | {change} | rdb_changes_since_last_sa | — | The number of writes since the last successful save. |

### Publish/subscribe

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.pubsub.channels | Vended | UpDownCounter | {channel} | pubsub_channels | — | The number of active global publish/subscribe channels. |
| valkey.pubsub.shard_channels | Vended | UpDownCounter | {channel} | pubsubshard_channels | — | The number of active sharded publish/subscribe channels. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.pubsub.patterns | Detailed only | UpDownCounter | {patter | pubsub_patterns | — | The number of active publish/subscribe pattern subscriptions. |

### Socket I/O

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.io.reads_processed | Detailed only | Counter | {event} | total_reads_processed | — | The number of socket read events processed by the event loop. |
| valkey.io.writes_processed | Detailed only | Counter | {event} | total_writes_processed | — | The number of socket write events processed by the event loop. |
| valkey.io.threaded.reads_processed | Detailed only | Counter | {event} | io_threaded_reads_process | — | The number of client reads handled by I/O threads. |
| valkey.io.threaded.writes_processed | Detailed only | Counter | {event} | io_threaded_writes_processed | — | The number of client writes handled by I/O threads. |
| valkey.io.threaded | Detailed only | Counter | {batch} | io_threaded_tota | — | The number of command prefetch batches executed by I/O threads. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| .prefetch_batches | | | | prefetc batches | | |
| valkey.io.threaded.prefetch_entries | Detailed only | Counter | {entry} | io_thre ed_tota prefetc entries | — | The number of entries prefetched by I/O threads. |

The valkey.io.threaded.* metrics are emitted only when I/O threading is enabled.

**Event loop**

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.eventloop.cycles | Detailed only | Counter | {cycle} | eventlo _cycles | — | The total number of event loop iterations. |
| valkey.eventloop.duration | Detailed only | Counter | s | eventlo _durati _sum | — | The total time spent in the event loop, including I/O and command processing. |
| valkey.eventloop.command.duration | Detailed only | Counter | s | eventlo _durati _cmd_su | — | The total time spent executing commands in the event loop. This is a subset of valkey.eventloop.duration . |

### Scripts and functions

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.scripts.cached | Detailed only | UpDownCounter | {script} | number_cached_scripts | — | The number of scripts in the script cache. |
| valkey.scripts.evicted | Detailed only | Counter | {script} | evicted_scripts | — | The number of scripts evicted from the script cache. |
| valkey.functions.count | Detailed only | UpDownCounter | {function} | number_functions | — | The number of registered functions. |
| valkey.functions.libraries.count | Detailed only | UpDownCounter | {library} | number_libraries | — | The number of loaded function libraries. |

### Client-side caching

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.tracking.items | Detailed only | UpDownCounter | {item} | tracking_total_items | — | The number of tracked items, which is the sum of the number of clients tracking each key. |

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.tracking.prefixes | Detailed only | UpDownCounter | {prefix} | tracking_total_prefixes | — | The number of prefixes tracked in broadcast mode. |

**Access control**

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| valkey.acl.denied | Vended | Counter | {request} | acl_acls_denied_<reason> | valkey.acl.denied.reason | The number of requests denied by access control, broken down by reason. |

The valkey.acl.denied.reason attribute takes one of the following values.

| Value | Description |
|---|---|
| auth | Authentication failures. |
| cmd | Commands denied by an access control list (ACL). |
| channel | Publish/subscribe channel accesses denied by an ACL. |
| key | Key accesses denied by an ACL. |
| tls_cert | TLS client certificate authentication failures. |

### Errors

The following metric is collected from the `errorstat_<prefix>` entries that `INFO` returns, where `<prefix>` is an error prefix such as `ERR` or `WRONGTYPE`.

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| `valkey.errors` | Vended | Counter | `{error}` | `errorstat_<prefix>:count` | `valkey.error.type` | The number of error replies, broken down by error type. |

The `valkey.error.type` attribute contains the error prefix that the cache engine returned, which is the first word of the error reply, for example `ERR`, `WRONGTYPE`, `NOAUTH`, or `OOM`. Because the attribute reports the prefix rather than the full message, each value represents a class of error rather than one specific error.

### Traffic management

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| `valkey.throttling.active` | Vended | Gauge | `{state}` | — | — | Set to 1 while traffic management is active, otherwise 0. Datapoints of 1 can indicate that the node is underscaled for the workload. |

### Durability

The following metrics are emitted for replication groups that use a durability configuration with a bounded recovery point objective (RPO).

| Metric | Tier | Type | Unit | INFO field | Additional attributes | Description |
|---|---|---|---|---|---|---|
| `valkey.durability.lag` | Vended | Gauge | `s` | — | — | The age of the oldest write that has not yet been persisted to the Multi-AZ transactional log. |
| `valkey.durability.buffer_exceeded_errors` | Vended | Counter | `{error}` | — | — | The number of writes rejected because the durability window was exceeded. Writes are rejected as `valkey.durability.lag` approaches 10 seconds. |

### Process

| Metric | Tier | Type | Unit | Additional attributes | Description |
|---|---|---|---|---|---|
| `process.cpu.time` | Vended | Counter | `s` | `cpu.mode`, `thread.type` | The CPU time consumed by the cache engine process. |
| `process.paging.faults` | Vended | Counter | `{fault}` | `system.paging.fault.type` | The number of major page faults for the cache engine |

| Metric | Tier | Type | Unit | Additional attributes | Description |
|---|---|---|---|---|---|
| | | | | | process. Minor faults are not reported. |

The additional attributes take the following values.

| Attribute | Values | Description |
|---|---|---|
| `cpu.mode` | `user` \| `system` | Whether the CPU time was spent in user mode or system mode. |
| `thread.type` | `main` | The type of thread that consumed the CPU time. Only the main thread is reported, so this metric does not include CPU time consumed by I/O threads. |
| `system.paging.fault.type` | `major` | The type of page fault. Only major faults are reported. |

**Host memory**

| Metric | Tier | Type | Unit | Additional attributes | Description |
|---|---|---|---|---|---|
| `system.memory.usage` | Vended | UpDownCounter | By | `system.memory.state = free` | The host memory, by state. The `free` state reports freeable memory. |

| Metric | Tier | Type | Unit | Additional attributes | Description |
|---|---|---|---|---|---|
| system.paging.usage | Vended | UpDownCounter | By | system.paging.state = used | The host swap space, by state. The used state reports swap space in use. |

### Host network

| Metric | Tier | Type | Unit | Additional attributes | Description |
|---|---|---|---|---|---|
| system.network.io | Vended | Counter | By | network.direction | The total bytes transferred on the host network interface. |
| system.network.packet.count | Vended | Counter | {packet} | network.direction | The total packets transferred on the host network interface. |
| system.network.io.rate | Vended | ExponentialHistogram | By/s | network.direction | The distribution of the per-second byte rate on the host network interface. The interface is sampled once a second, and each datapoint is the distribution of those per-second rates. |
| system.network.packet.rate | Vended | ExponentialHistogram | {packet}/s | network.direction | The distribution of the per-second packet rate on the host network interface. The interface is sampled once a |

| Metric | Tier | Type | Unit | Additional attributes | Description |
|---|---|---|---|---|---|
| | | | | | second, and each datapoint is the distribution of those per-second rates. |
| system.network.allowance_exceeded | Vended | Counter | {packet} | network.allowance.exceeded.reason | The number of packets that were queued or dropped because a network allowance was exceeded, broken down by reason. The counters increment for each affected packet, not once for each time the allowance was exceeded. |
| system.network.bandwidth.limit | Vended | Gauge | By/s | — | The baseline network bandwidth for the node's instance type. The limit is the same in each direction and applies to each direction separately. |

The additional attributes take the following values.

| Attribute | Values | Description |
|---|---|---|
| network.io.direction | receive \| transmit | The direction of the network traffic. |
| network.allowance_exceeded.reason | bandwidth_in \| bandwidth | The allowance that was exceeded: inbound bandwidth, outbound bandwidth, packets per second, or tracked connections. |

| Attribute | Values | Description |
| --- | --- | --- |
| | `_out \|pps\| conntrack` | |

## Example queries for common monitoring tasks

The following queries answer common questions about a node-based Valkey replication group. Each one is ready to copy into a CloudWatch dashboard, and you can edit any of them to suit your own workload.

Every query is grouped by topic and collapsed. Expand an entry to see the query it contains.

Some queries use metrics that are emitted only while they are selected in a resource metrics configuration. These queries are marked **Requires detailed monitoring**, and return no data until you select the metrics that they name. For more information, see Turning on detailed monitoring.

> **Note**
>
> These queries assume that you understand the PromQL constructs they use. For an introduction to each construct, with worked examples, see Querying your metrics.

### Substituting your own values

Replace the following placeholders with values from your own replication group.

| Placeholder | Replace with |
| --- | --- |
| `my-cluster` | Your replication group ID |
| `my-cluster-0001-001` | A node ID |
| `prod-.*` | A regular expression that matches the replication group IDs you want |

**Note**

A query can return at most 500 time series. Queries that break results down by node and by command can exceed that limit on a large replication group, in which case the response is truncated. Aggregate the result, or narrow it with a more specific label matcher.

**Note**

The ranges in these queries, such as `[5m]`, suit a metric emitted every 60 seconds. If you have selected a metric, it is emitted every 15 seconds and you can shorten its ranges to `[1m]` for a more responsive result.

## Turning a query into an alarm

A PromQL alarm carries its threshold inside the query. To turn any of these queries into an alarm, append a comparison. The query then returns a series only while the condition is true, and the alarm tracks each returned series separately. For example, the following query returns a series for each node that is using more than 90 percent of its memory limit.

```
100 * {"valkey.memory.used", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
    / {"valkey.memory.max",  "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
    > 90
```

Choose a threshold that suits your workload. A query that already ends in a comparison, such as `== 0`, can be used as an alarm unchanged. For how a PromQL alarm evaluates a query, see Creating alarms from queries.

## Memory and capacity

### Memory used as a percentage of the maxmemory limit

```
100 * {"valkey.memory.used", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
    / {"valkey.memory.max",  "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
```

A ratio against the configured limit, so it is comparable across different configurations.

**When memory is forecast to reach the limit**

```
100 * predict_linear({"valkey.memory.used",
                      "@resource.aws.elasticache.replication_group.id"="my-cluster"}
[1h], 3600)
    / on ("@resource.aws.elasticache.node.id")
      {"valkey.memory.max", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}
```

`predict_linear` fits the trend of the last hour and projects it one hour ahead, so a value over 100 means that the node is projected to reach `maxmemory` within the hour. A linear fit does not model a workload that grows in steps.

**Memory counted toward eviction, as a percentage of the limit**

```
100 * ( {"valkey.memory.used",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
      - {"valkey.memory.not_counted_for_evict",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"} )
    / {"valkey.memory.max",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
```

The engine excludes some memory when deciding whether to evict, so this tracks the eviction decision more closely than total memory use does.

**Memory overhead ratio**

**Requires detailed monitoring:** `valkey.memory.used.overhead`

```
100 * {"valkey.memory.used.overhead",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
    / {"valkey.memory.used",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
```

A high share usually points to client buffer growth or replication backlog rather than dataset size.

**Peak memory over a window**

```
max_over_time({"valkey.memory.used",
               "@resource.aws.elasticache.replication_group.id"="my-cluster"}[24h])
```

The range sets the period, up to 7 days.

### How fast the dataset is growing

```
deriv({"valkey.memory.used",
       "@resource.aws.elasticache.replication_group.id"="my-cluster"}[30m])
```

`deriv` is used because `valkey.memory.used` can decrease as well as increase, so `rate` does not apply.

## CPU and compute

### Engine CPU as a percentage of one vCPU

```
100 * sum without ("cpu.mode") (
        rate({"process.cpu.time",
              "thread.type"="main",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

The main thread is single-threaded, so approaching 100 percent means the engine is saturated regardless of how many vCPUs the node has.

### User vs system CPU split

```
100 * sum by ("cpu.mode", "@resource.aws.elasticache.node.id") (
        rate({"process.cpu.time",
              "thread.type"="main",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### How fast swap is growing

```
deriv({"system.paging.usage",
       "system.paging.state"="used",
       "@resource.aws.elasticache.replication_group.id"="my-cluster"}[30m])
```

A sustained positive value means the host is still moving pages to swap, which is worse than a steady non-zero figure.

### Host page-fault rate

```
rate({"process.paging.faults",
```

```
      "system.paging.fault.type"="major",
      "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

Only major faults are reported. They require disk access, so a sustained rate indicates memory pressure.

## Throughput and commands

### Total command rate per node

```
rate({"valkey.commands.processed",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

### Read vs write command rate

```
sum by ("valkey.command.type") (
  rate({"valkey.command.calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### Write ratio

```
100 * sum(rate({"valkey.command.calls", "valkey.command.type"="write",
                "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / sum(rate({"valkey.command.calls", "valkey.command.type"=~"read|write",
                "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

Commands that neither read nor write, such as `PING`, carry no `valkey.command.type`, so the denominator counts reads and writes only.

### Top 10 commands by call rate

```
topk(10, sum by ("valkey.command") (
  rate({"valkey.command.calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])))
```

### Call rate per individual command

```
sum by ("valkey.command") (
  rate({"valkey.command.calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### Call rate for a specific set of commands

```
sum by ("valkey.command") (
  rate({"valkey.command.calls",
        "valkey.command"=~"GET|MGET|HGET|LRANGE",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### Command rate by category

```
sum by ("valkey.command.category") (
  rate({"valkey.command.calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### Rate for one data type

```
sum(rate({"valkey.command.calls", "valkey.command.category"="sortedset",
          "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### TPS of each shard

```
sum by ("@resource.aws.elasticache.shard.id") (
  rate({"valkey.commands.processed",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

Compare shards against each other to find imbalance.

### Each node's share of its shard's traffic

```
  rate({"valkey.commands.processed",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
/ on ("@resource.aws.elasticache.shard.id") group_left
  sum by ("@resource.aws.elasticache.shard.id") (
    rate({"valkey.commands.processed",
          "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

## Latency (per command)

### Average latency per command, in microseconds

```
1e6 * sum by ("valkey.command") (
```

```
        rate({"valkey.command.duration",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / (sum by ("valkey.command") (
        rate({"valkey.command.calls",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])) > 0)
```

1e6 converts seconds to microseconds. The result is an average over the range.

### Slowest commands right now

```
topk(5,
  1e6 * sum by ("valkey.command") (rate({"valkey.command.duration",
            "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
      / (sum by ("valkey.command") (rate({"valkey.command.calls",
            "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])) > 0))
```

### Average read latency vs average write latency

```
1e6 * sum by ("valkey.command.type") (
        rate({"valkey.command.duration",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / (sum by ("valkey.command.type") (
        rate({"valkey.command.calls",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])) > 0)
```

### Average latency by command category

```
1e6 * sum by ("valkey.command.category") (
        rate({"valkey.command.duration",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / (sum by ("valkey.command.category") (
        rate({"valkey.command.calls",
              "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])) > 0)
```

### Aggregate average command latency across the replication group

```
1e6 * sum(rate({"valkey.command.duration",
                "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / sum(rate({"valkey.command.calls",
                "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

## Cache effectiveness

### Keyspace hit ratio

```
100 * sum(rate({"valkey.keyspace.hits",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
    / (sum(rate({"valkey.keyspace.hits",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
     +
 sum(rate({"valkey.keyspace.misses","@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m])))
```

### Eviction rate

```
rate({"valkey.keys.evicted", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m])
```

Keys are evicted only under an eviction policy. Under `noeviction`, writes are rejected instead.

### Expiration rate

```
rate({"valkey.keys.expired", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m])
```

### Per-field (hash TTL) expiration rate

```
rate({"valkey.keys.expired_fields",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

## Connections and clients

### Connection utilization vs maxclients

```
100 * {"valkey.clients.connected",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
    / {"valkey.clients.max",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
```

A ratio against the configured limit, so it is comparable across different configurations.

### Rejected-connection rate

```
rate({"valkey.connections.rejected",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

### Connection acceptance rate

```
rate({"valkey.connections.received",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

A high rate can indicate that clients are not pooling connections.

## Client buffer pressure

### Clients disconnected for exceeding the output buffer limit

**Requires detailed monitoring:** `valkey.clients.output_buffer_disconnections`

```
rate({"valkey.clients.output_buffer_disconnections",
      "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

A disconnection is data the client never received.

### Clients disconnected for exceeding the query buffer limit

**Requires detailed monitoring:** `valkey.clients.query_buffer_disconnections`

```
rate({"valkey.clients.query_buffer_disconnections",
      "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

A disconnection is a request that was never completed.

### Clients evicted due to the client-memory limit

**Requires detailed monitoring:** `valkey.clients.evicted`

```
rate({"valkey.clients.evicted", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m])
```

Eviction means client memory reached its configured cap.

## Replication and durability

### Replica link down

```
{"valkey.replication.link_healthy", "valkey.role"="replica",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"} == 0
```

Returns a series only for a replica whose link is down, so it can be used as an alarm unchanged.

### Sync in progress

**Requires detailed monitoring:** `valkey.replication.sync.in_progress`

```
{"valkey.replication.sync.in_progress",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"} == 1
```

### Write throughput on each primary

```
deriv({"valkey.replication.offset", "valkey.role"="primary",
       "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

Reported in bytes per second.

### Durability buffer-exceeded errors

```
rate({"valkey.durability.buffer_exceeded_errors",
      "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

Every occurrence is a rejected write.

## Persistence

### RDB save in progress

```
{"valkey.persistence.rdb.save_in_progress",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"} == 1
```

Expected during backups and diskless replication.

## Network

### Network utilization vs the instance baseline

```
100 * rate({"system.network.io",
            "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
    / on ("@resource.aws.elasticache.node.id") group_left
      {"system.network.bandwidth.limit",
       "@resource.aws.elasticache.replication_group.id"="my-cluster"}
```

A ratio against the limit the instance reports, so it is comparable across node types. The limit applies to each direction separately, so each direction is compared against the whole limit.

### Inbound utilization only

```
100 * rate({"system.network.io", "network.io.direction"="receive",
            "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
    / on ("@resource.aws.elasticache.node.id")
      {"system.network.bandwidth.limit",
       "@resource.aws.elasticache.replication_group.id"="my-cluster"}
```

Inbound traffic is measured against the whole limit, not half of it.

### Network allowance exceedance by reason

```
sum by ("network.allowance_exceeded.reason") (
  rate({"system.network.allowance_exceeded",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

An exceedance means packets were queued or dropped.

### Inbound/outbound throughput per node

```
rate({"system.network.io", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m])
```

To compare against the limit, use Network utilization vs the instance baseline.

### Peak network utilization vs the baseline, per direction

```
100 * histogram_quantile(0.99,
```

```
       {"system.network.io.rate",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"})
   / on ("@resource.aws.elasticache.node.id") group_left
     {"system.network.bandwidth.limit",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}
```

The p99 of the distribution catches bursts that an averaged value smooths away.

**p99 inbound byte-rate (micro-burst detection)**

```
histogram_quantile(0.99,
  {"system.network.io.rate", "network.io.direction"="receive",
   "@resource.aws.elasticache.replication_group.id"="my-cluster"})
```

Reveals one-second bursts that an average hides.

**p99 packet rate**

```
histogram_quantile(0.99,
  {"system.network.packet.rate", "network.io.direction"="receive",
   "@resource.aws.elasticache.replication_group.id"="my-cluster"})
```

**Packet rate per node, per direction**

```
rate({"system.network.packet.count",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

Packet rate limits are separate from bandwidth limits. Exceeding one is reported as a `pps` allowance exceedance.

**Replication throughput of each primary**

**Requires detailed monitoring:** `valkey.network.output.replication`

```
sum by ("@resource.aws.elasticache.node.id") (
  rate({"valkey.network.output.replication", "valkey.role"="primary",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

This counts the bytes sent to every replica, so it scales with the number of replicas. For write throughput, see Write throughput on each primary.

### Replication share of network

**Requires detailed monitoring:** `valkey.network.output`, `valkey.network.output.replication`

```
rate({"valkey.network.output.replication",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
/
rate({"valkey.network.output",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

### Inbound engine traffic

**Requires detailed monitoring:** `valkey.network.input`

```
rate({"valkey.network.input", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[1m])
```

Compare against host-level network metrics to separate engine traffic from total traffic.

## Errors and rejections

### Error reply rate by error code

```
sum by ("valkey.error.type") (
  rate({"valkey.errors", "@resource.aws.elasticache.replication_group.id"="my-cluster"}
[5m]))
```

### Top error codes

```
topk(5, sum by ("valkey.error.type") (
  rate({"valkey.errors", "@resource.aws.elasticache.replication_group.id"="my-cluster"}
[5m])))
```

### Rejected-call rate per command

**Requires detailed monitoring:** `valkey.command.rejected_calls`

```
sum by ("valkey.command") (
  rate({"valkey.command.rejected_calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### Failed-call rate per command

**Requires detailed monitoring:** `valkey.command.failed_calls`

```
sum by ("valkey.command") (
  rate({"valkey.command.failed_calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

### Failed calls by command category

**Requires detailed monitoring:** `valkey.command.failed_calls`

```
sum by ("valkey.command.category") (
  rate({"valkey.command.failed_calls",
        "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

## Access control and security

### Access-denied rate by reason

```
sum by ("valkey.acl.denied.reason") (
  rate({"valkey.acl.denied", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m]))
```

A denial is either a misconfigured client or an unauthorized attempt.

### Authentication failure rate

```
sum(rate({"valkey.acl.denied", "valkey.acl.denied.reason"="auth",
          "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

Sustained failures indicate either a credential problem or a brute-force attempt.

## Event loop

### Event-loop utilization

**Requires detailed monitoring:** `valkey.eventloop.duration`

```
rate({"valkey.eventloop.duration",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

The fraction of time that the main thread is busy. The event loop is the engine's serialization point, so latency rises sharply as this approaches 1.

**Share of event-loop time spent executing commands**

**Requires detailed monitoring:** `valkey.eventloop.command.duration`, `valkey.eventloop.duration`

```
rate({"valkey.eventloop.command.duration",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
/
rate({"valkey.eventloop.duration",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

A falling share points to engine overhead such as replication or defragmentation.

**Average event-loop cycle time (microseconds)**

**Requires detailed monitoring:** `valkey.eventloop.cycles`, `valkey.eventloop.duration`

```
1e6 * rate({"valkey.eventloop.duration",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
    / rate({"valkey.eventloop.cycles",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

1e6 converts seconds to microseconds.

**Event-loop cycle rate**

**Requires detailed monitoring:** `valkey.eventloop.cycles`

```
rate({"valkey.eventloop.cycles", "@resource.aws.elasticache.replication_group.id"="my-
cluster"}[5m])
```

Read with the average cycle time: a falling cycle rate together with rising cycle duration is the signature of a stall. An idle node still shows a floor from timer activity.

**Event-loop time not spent on CPU**

**Requires detailed monitoring:** `valkey.eventloop.duration`

```
  rate({"valkey.eventloop.duration",
 "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m])
```

```
- on ("@resource.aws.elasticache.node.id")
  sum without ("cpu.mode") (
    rate({"process.cpu.time", "thread.type"="main",
          "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

Event-loop duration is wall-clock time and CPU time is time on the processor, so the two normally track each other. A sustained gap means that the main thread is busy but waiting on something other than CPU, such as swap or a fork for a background save.

## Fleet-wide and cross-resource patterns

### Top 10 nodes by memory across every prod replication group

```
topk(10, {"valkey.memory.used",
          "@resource.aws.elasticache.replication_group.id"=~"prod-.*"})
```

### Command rate rolled up by replication group

```
sum by ("@resource.aws.elasticache.replication_group.id") (
  rate({"valkey.commands.processed",
        "@resource.aws.elasticache.replication_group.id"=~"prod-.*"}[5m]))
```

### Highest-error clusters in the fleet

```
topk(5, sum by ("@resource.aws.elasticache.replication_group.id") (
  rate({"valkey.errors",
        "@resource.aws.elasticache.replication_group.id"=~"prod-.*"}[5m])))
```

### Network throughput rolled up by Availability Zone

```
sum by ("@resource.cloud.availability_zone_id", "network.io.direction") (
  rate({"system.network.io",
 "@resource.aws.elasticache.replication_group.id"=~"prod-.*"}[5m]))
```

### Write throughput of each shard

```
sum by ("@resource.aws.elasticache.shard.id") (
  deriv({"valkey.replication.offset", "valkey.role"="primary",
         "@resource.aws.elasticache.replication_group.id"="my-cluster"}[5m]))
```

Compare with TPS of each shard to see whether write traffic and total traffic are balanced the same way.

# Metrics and events for Memcached caches and clusters

This section describes the metrics and events that you can monitor when working with Memcached node-based and serverless caches.

**Topics**


- Metrics for ElastiCache Serverless for Memcached
- Events for ElastiCache Serverless for Memcached
- Metrics for node-based Memcached clusters
- Events for node-based Memcached clusters


## Metrics for ElastiCache Serverless for Memcached

This section describes the metrics and events that you can monitor when working with ElastiCache Serverless for Memcached.

The `AWS/ElastiCache` namespace includes the following CloudWatch metrics for your Memcached serverless caches.

| Metric | Description | Unit |
|---|---|---|
| BytesUsedForCache | The total number of bytes used by the data stored in your cache. | Bytes |
| ElastiCacheProcessingUnits | The total number of ElastiCacheProcessingUnits (ECPUs) consumed by the requests executed on your cache | Count |
| SuccessfulReadRequestLatency | Latency of successful read requests. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| SuccessfulWriteRequestLatency | Latency of successful write requests | Microseconds |
| TotalCmdsCount | Total count of all commands executed on your cache | Count |
| CurrConnections | The number of client connections to your cache. | Count |
| ThrottledCmds | The number of requests that were throttled by ElastiCache because the workload was scaling faster than ElastiCache can scale. | Count |
| NewConnections | The total number of connections that have been accepted by the server during this period. | Count |
| CurrItems | The number of items in the cache. | Count |
| NetworkBytesIn | Total bytes transferred in to cache | Bytes |
| NetworkBytesOut | Total bytes transferred out of cache | Bytes |
| Evictions | The count of keys evicted by the cache | Count |
| Reclaimed | The number of keys expired by the cache | Count |

### Command level metrics

ElastiCache also emits the following Memcached command level metrics

| Metric | Description | Unit |
| --- | --- | --- |
| CmdGet | The number of get commands the cache has received. | Count |
| CmdSet | The number of set commands the cache has received. | Count |
| CmdTouch | The number of touch commands the cache has received. | Count |
| GetHits | The number of get requests the cache has received where the key requested was found. | Count |
| GetMisses | The number of get requests the cache has received where the key requested was not found. | Count |
| IncrHits | The number of increment requests the cache has received where the key requested was found. | Count |
| IncrMisses | The number of increment requests the cache has received where the key requested was not found. | Count |
| DecrHits | The number of decrement requests the cache has received where the requested key was found. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| DecrMisses | The number of decrement requests the cache has received where the requested key was not found. | Count |
| DeleteHits | The number of delete requests the cache has received where the requested key was found. | Count |
| DeleteMisses | The number of delete requests the cache has received where the requested key was not found. | Count |
| TouchHits | The number of keys that have been touched and were given a new expiration time. | Count |
| TouchMisses | The number of keys that have been touched, but were not found. | Count |
| CasHits | The number of cas requests the cache has received where the requested key was found and the cas value matched. | Count |
| CasMisses | The number of cas requests the cache has received where the key requested was not found. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CasBadval | The number of cas requests the cache has received where the cas value did not match the cas value stored. | Count |
| CmdFlush | The number of flush commands the cache has received. | Count |

## Events for ElastiCache Serverless for Memcached

ElastiCache logs events that relate to your serverless cache. This information includes the date and time of the event, the source name and source type of the event, and a description of the event. You can easily retrieve events from the log using the ElastiCache console, the AWS CLI describe-events command, or the ElastiCache API action `DescribeEvents`.

You can choose to monitor, ingest, transform, and act on ElastiCache events using Amazon EventBridge. Learn more in the Amazon EventBridge getting started guide.

### Viewing ElastiCache events (Console)

To view events using the ElastiCache console:

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/
2. To see a list of all available events, in the navigation pane, choose **Events**.
3. On the *Events* screen each row of the list represents one event and displays the event source, the event type, the GMT time of the event, and a description of the event. Using the **Filter** you can specify whether you want to see all events, or just events of a specific type in the event list.

### Viewing ElastiCache events (AWS CLI)

To generate a list of ElastiCache events using the AWS CLI, use the command describe-events. You can use optional parameters to control the type of events listed, the time frame of the events listed, the maximum number of events to list, and more.

The following code lists up to 40 serverless cache events.

```
aws elasticache describe-events --source-type serverless-cache --max-items 40
```

The following code lists all events for serverless caches for the past 24 hours (1440 minutes).

```
aws elasticache describe-events --source-type serverless-cache --duration 1440
```

### Serverless Events

This section documents the different types of events that you may receive for your serverless caches.

### Serverless Cache Creation Events

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| Cache created | Cache arn | creation | serverless-cache | Cache <cache-name> is created and ready to use. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Insufficient free IP addresses to create VPC endpoint. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Invalid subnets provided in the request. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. Quota |

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| | | | | limit reached for creating VPC endpoint. |
| Cache creation failed | Cache arn | failure | serverless-cache | Failed to create cache <cache-name>. You do not have permissions to create a VPC endpoint. |

**Serverless Cache Update Events (Memcached)**

| Detail-Type | Resources list | Category | Source | Message |
|---|---|---|---|---|
| Cache updated | Cache arn | configuration change | serverless-cache | SecurityGroups updated for the cache <cache-name>. |
| Cache updated | Cache arn | configuration change | serverless-cache | Tags updated for the cache <cache-name>. |
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. SecurityGroups update failed. |
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. SecurityGroups update |

| Detail-Type | Resources list | Category | Source | Message |
|---|---|---|---|---|
| | | | | failed because of insufficient permissions. |
| Cache updated failed | Cache arn | configuration change | serverless-cache | An update to the cache <cache-name> failed. SecurityGroups update failed because the SecurityGroups are invalid. |

**Serverless Cache Deletion Events (Memcached)**

| Detail-Type | Resources list | Category | Source | Message |
|---|---|---|---|---|
| Cache deleted | Cache arn | deletion | serverless-cache | Cache <cache-name> was deleted. |

**Serverless Cache Usage Limit Events (Memcached)**

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| Cache updated | Cache arn | configuration change | serverless-cache | Limits updated for the cache <cache-name>. |
| Cache updated failed | Cache arn | failure | serverless-cache | A limit update to the cache <cache-name> failed because |

| Detail-Type | Description | Unit | Source | Message |
|---|---|---|---|---|
| | | | | the cache was deleted. |
| Cache updated failed | Cache arn | failure | serverless-cache | A limit update to the cache <cache-name> failed due to invalid configuration. |

**Serverless Cache Snapshot Events (Memcached)**

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot created | Cache arn<br>Snapshot arn | creation | serverless-cache-snapshot | Snapshot <snapshot-name> created for cache <cache-name>. |
| Snapshot creation failed | Cache arn<br>Snapshot arn | failure | serverless-cache-snapshot | Failed to create snapshot for cache <cache-name>. Snapshot <snapshot-name> creation failed with Customer Managed Key <key-id> <reason>.<br>Failure reason messages: |

| Detail-Type | Resources-list | Category | Source | Message |
| --- | --- | --- | --- | --- |
| | | | | • because Customer Managed Key is disabled<br>• because Customer Managed Key cannot be found<br>• because the request timed out |
| Snapshot creation failed | Cache arn<br><br>Snapshot arn | failure | serverless-cache-snapshot | Failed to create snapshot for cache <cache-name>. Snapshot <snapshot-name> creation failed because <reason>.<br><br>Default reason:<br><br>• because of an internal error |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket %s because ElastiCache does not have permissions to the bucket. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because there is already an object with the same name in the bucket. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because bucket owner account Id has changed. |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because the S3 bucket is not accessible. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because the bucket is not accessible. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' because bucket does not exist. |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s' with source snapshot Customer Managed Key %s <reason>. |
| Snapshot export failed | Snapshot arn | failure | serverless-cache-snapshot | Failed to export snapshot for cache <cache-name>. Could not export snapshot to bucket '%s'. |
| Snapshot copy failed | Snapshot arn-1<br>Snapshot arn-2 | failure | serverless-cache-snapshot | Failed to copy snapshot <snapshot-name>. Could not copy snapshot '%s' to snapshot '%s' with source snapshot Customer Managed Key <key-id> <reason-name>. |

| Detail-Type | Resources-list | Category | Source | Message |
|---|---|---|---|---|
| Snapshot copy failed | Snapshot arn-1<br>Snapshot arn-2 | failure | serverless-cache-snapshot | Failed to copy snapshot <snapshot-name>. Could not copy snapshot '%s' to snapshot '%s' with target snapshot Customer Managed Key '%s' '%s'. |

## Metrics for node-based Memcached clusters

This section describes the CloudWatch metrics that you can monitor when working with node-based Memcached clusters. These metrics are measured per cache node in 60-second intervals.

**Host-level metrics**

| Metric | Description | Unit |
|---|---|---|
| `CPUUtilization` | Percentage of CPU utilization for the entire host. Because Memcached is multi-thr eaded, this metric can be as high as 90%. If you exceed this threshold, scale your cluster up by using a larger cache node type, or scale out by adding more cache nodes. | Percent |
| `FreeableMemory` | Amount of free memory available on the host, derived from RAM, buffers, and | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| | cache that the OS reports as freeable. | |
| NetworkBytesIn | Number of bytes the host has read from the network. | Bytes |
| NetworkBytesOut | Number of bytes sent out on all network interfaces by the instance. | Bytes |
| NetworkPacketsIn | Number of packets received on all network interfaces by the instance. | Count |
| NetworkPacketsOut | Number of packets sent out on all network interfaces by the instance. | Count |
| SwapUsage | Amount of swap used on the host. | Bytes |

### Memcached metrics

| Metric | Description | Unit |
| --- | --- | --- |
| BytesReadIntoMemcached | Number of bytes read from the network by the cache node. | Bytes |
| BytesUsedForCacheItems | Number of bytes used to store cache items. | Bytes |
| BytesWrittenOutFromMemcached | Number of bytes written to the network by the cache node. | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| CasBadval | Number of CAS requests where the Cas value did not match. | Count |
| CasHits | Number of Cas requests where the key was found and the Cas value matched. | Count |
| CasMisses | Number of Cas requests where the key was not found. | Count |
| CmdFlush | Number of flush commands received. | Count |
| CmdGet | Number of get commands received. | Count |
| CmdSet | Number of set commands received. | Count |
| CurrConnections | Number of connections connected to the cache at an instant in time. | Count |
| CurrItems | Number of items currently stored in the cache. | Count |
| DecrHits | Number of decrement requests where the key was found. | Count |
| DecrMisses | Number of decrement requests where the key was not found. | Count |
| DeleteHits | Number of delete requests where the key was found. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| DeleteMisses | Number of delete requests where the key was not found. | Count |
| Evictions | Number of non-expired items evicted to allow space for new writes. | Count |
| GetHits | Number of get requests where the key was found. | Count |
| GetMisses | Number of get requests where the key was not found. | Count |
| IncrHits | Number of increment requests where the key was found. | Count |
| IncrMisses | Number of increment requests where the key was not found. | Count |
| NewConnections | Number of new connections the cache has received. | Count |
| NewItems | Number of new items the cache has stored. | Count |
| Reclaimed | Number of expired items evicted to allow space for new writes. | Count |
| UnusedMemory | Amount of memory not used by data. | Bytes |

## Events for node-based Memcached clusters

ElastiCache sends notifications for significant cluster events using Amazon Simple Notification Service. You can monitor events using the ElastiCache console, the AWS CLI `describe-events` command, or the ElastiCache API `DescribeEvents` action.

To view events using the AWS CLI, use the `--source-type cache-cluster` parameter.

The following examples show how to use the AWS CLI to list cache cluster events:

List up to 40 cache cluster events:

```
aws elasticache describe-events --source-type cache-cluster --max-items 40
```

List cache cluster events for the past 24 hours:

```
aws elasticache describe-events --source-type cache-cluster --duration 1440
```

For more information about managing Amazon SNS notifications for events, see the Amazon SNS event monitoring topic.

# Logging Amazon ElastiCache API calls with AWS CloudTrail

Amazon ElastiCache is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon ElastiCache. CloudTrail captures all API calls for Amazon ElastiCache as events, including calls from the Amazon ElastiCache console and from code calls to the Amazon ElastiCache API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon ElastiCache. 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 Amazon ElastiCache, 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.

## Amazon ElastiCache information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon ElastiCache, 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 Amazon ElastiCache, 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 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 Amazon ElastiCache actions are logged by CloudTrail and are documented in the ElastiCache API Reference. For example, calls to the `CreateCacheCluster`, `DescribeCacheCluster` and `ModifyCacheCluster` 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 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 Amazon ElastiCache 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 are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateCacheCluster` action.


```
{
    "eventVersion":"1.01",
    "userIdentity":{
        "type":"IAMUser",
        "principalId":"EXAMPLEEXAMPLEEXAMPLE",
        "arn":"arn:aws:iam::123456789012:user/elasticache-allow",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"elasticache-allow"
    },
    "eventTime":"2014-12-01T22:00:35Z",
    "eventSource":"elasticache.amazonaws.com",
    "eventName":"CreateCacheCluster",
    "awsRegion":"us-west-2",
    "sourceIPAddress":"192.0.2.01",
    "userAgent":"AWS CLI/ElastiCache 1.10 API 2014-12-01",
    "requestParameters":{
        "numCacheNodes":2,
        "cacheClusterId":"test-memcached",
        "engine":"memcached",
        "aZMode":"cross-az",
        "cacheNodeType":"cache.m5.large",

    },
    "responseElements":{
        "engine":"memcached",
        "clientDownloadLandingPage":"https://console.aws.amazon.com/elasticache/
home#client-download:",
        "cacheParameterGroup":{
            "cacheParameterGroupName":"default.memcached1.4",
            "cacheNodeIdsToReboot":{
            },
            "parameterApplyStatus":"in-sync"
        },
        "preferredAvailabilityZone":"Multiple",
        "numCacheNodes":2,
        "cacheNodeType":"cache.m5.large",

        "cacheClusterStatus":"creating",
        "autoMinorVersionUpgrade":true,
```

```
        "preferredMaintenanceWindow":"thu:05:00-thu:06:00",
        "cacheClusterId":"test-memcached",
        "engineVersion":"1.4.14",
        "cacheSecurityGroups":[
            {
                "status":"active",
                "cacheSecurityGroupName":"default"
            }
        ],
        "pendingModifiedValues":{
        }
    },
    "requestID":"104f30b3-3548-11e4-b7b8-6d79ffe84edd",
    "eventID":"92762127-7a68-42ce-8787-927d2174cde1"
}
```

The following example shows a CloudTrail log entry that demonstrates the `DescribeCacheCluster` action. Note that for all Amazon ElastiCache Describe calls (`Describe*`), the `ResponseElements` section is removed and appears as `null`.

```
{
    "eventVersion":"1.01",
    "userIdentity":{
        "type":"IAMUser",
        "principalId":"EXAMPLEEXAMPLEEXAMPLE",
        "arn":"arn:aws:iam::123456789012:user/elasticache-allow",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"elasticache-allow"
    },
    "eventTime":"2014-12-01T22:01:00Z",
    "eventSource":"elasticache.amazonaws.com",
    "eventName":"DescribeCacheClusters",
    "awsRegion":"us-west-2",
    "sourceIPAddress":"192.0.2.01",
    "userAgent":"AWS CLI/ElastiCache 1.10 API 2014-12-01",
    "requestParameters":{
        "showCacheNodeInfo":false,
        "maxRecords":100
    },
    "responseElements":null,
    "requestID":"1f0b5031-3548-11e4-9376-c1d979ba565a",
    "eventID":"a58572a8-e81b-4100-8e00-1797ed19d172"
```

```
}
```

The following example shows a CloudTrail log entry that records a `ModifyCacheCluster` action.

```
{
    "eventVersion":"1.01",
    "userIdentity":{
        "type":"IAMUser",
        "principalId":"EXAMPLEEXAMPLEEXAMPLE",
        "arn":"arn:aws:iam::123456789012:user/elasticache-allow",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"elasticache-allow"
    },
    "eventTime":"2014-12-01T22:32:21Z",
    "eventSource":"elasticache.amazonaws.com",
    "eventName":"ModifyCacheCluster",
    "awsRegion":"us-west-2",
    "sourceIPAddress":"192.0.2.01",
    "userAgent":"AWS CLI/ElastiCache 1.10 API 2014-12-01",
    "requestParameters":{
        "applyImmediately":true,
        "numCacheNodes":3,
        "cacheClusterId":"test-memcached"
    },
    "responseElements":{
        "engine":"memcached",
        "clientDownloadLandingPage":"https://console.aws.amazon.com/elasticache/
home#client-download:",
        "cacheParameterGroup":{
            "cacheParameterGroupName":"default.memcached1.4",
            "cacheNodeIdsToReboot":{
            },
            "parameterApplyStatus":"in-sync"
        },
        "cacheClusterCreateTime":"Dec 1, 2014 10:16:06 PM",
        "preferredAvailabilityZone":"Multiple",
        "numCacheNodes":2,
        "cacheNodeType":"cache.m5.large",
        "cacheClusterStatus":"modifying",
        "autoMinorVersionUpgrade":true,
        "preferredMaintenanceWindow":"thu:05:00-thu:06:00",
        "cacheClusterId":"test-memcached",
```

```
        "engineVersion":"1.4.14",
        "cacheSecurityGroups":[
            {
                "status":"active",
                "cacheSecurityGroupName":"default"
            }
        ],
        "configurationEndpoint":{
            "address":"test-memcached.example.cfg.use1prod.cache.amazonaws.com",
            "port":11211
        },
        "pendingModifiedValues":{
            "numCacheNodes":3
        }
    },
    "requestID":"807f4bc3-354c-11e4-9376-c1d979ba565a",
    "eventID":"e9163565-376f-4223-96e9-9f50528da645"
}
```


# Amazon SNS monitoring of ElastiCache events

When significant events happen for a cluster, ElastiCache sends notification to a specific Amazon SNS topic. Examples include a failure to add a node, success in adding a node, the modification of a security group, and others. By monitoring for key events, you can know the current state of your clusters and, depending upon the event, be able to take corrective action.

**Topics**

- Managing ElastiCache Amazon SNS notifications
- Viewing ElastiCache events
- Event Notifications and Amazon SNS

## Managing ElastiCache Amazon SNS notifications

You can configure ElastiCache to send notifications for important cluster events using Amazon Simple Notification Service (Amazon SNS). In these examples, you will configure a cluster with the Amazon Resource Name (ARN) of an Amazon SNS topic to receive notifications.

**Note**

- This topic assumes that you've signed up for Amazon SNS and have set up and subscribed to an Amazon SNS topic. For information on how to do this, see the Amazon Simple Notification Service Developer Guide.
- By default, the `API modify-replication-group` affects all groups in a Region and not just the current specified group. If you want to configure one specific group in a Region differently than the other groups, you can use the `--notification-topic-arn` option to create a separate topic for that group.

**Encrypted Amazon SNS topics are not supported**

ElastiCache cannot publish to an encrypted Amazon SNS topic, whether the topic uses the AWS managed key for Amazon SNS or a customer managed key. You can attach an encrypted topic to a cluster, and its status shows as active. When ElastiCache sends the first notification, it sets the topic status to inactive and does not emit an event. Use an unencrypted topic.

## Prerequisites

Before you configure Amazon SNS notifications for ElastiCache, verify the following requirements:

- The Amazon SNS topic must be in the same AWS Region as your ElastiCache cluster.
- The Amazon SNS topic must be owned by the same AWS account as your ElastiCache cluster.
- The Amazon SNS topic must be unencrypted. ElastiCache cannot publish to a topic that is encrypted with the AWS managed key for Amazon SNS or with a customer managed key.
- You must have permissions to modify the Amazon SNS topic access policy (`sns:SetTopicAttributes` or equivalent).

## Granting ElastiCache permission to publish to your Amazon SNS topic

To receive event notifications, you must grant ElastiCache permission to publish messages to your Amazon SNS topic. You do this by adding a resource-based policy to the Amazon SNS topic that allows the `elasticache.amazonaws.com` service principal to publish messages.

The following policy grants ElastiCache the required permission:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowElastiCachePublish",
      "Effect": "Allow",
      "Principal": {
        "Service": "elasticache.amazonaws.com"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:region:account-id:topic-name",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}
```

The following table describes the policy components.

| Component | Value | Description |
| --- | --- | --- |
| Principal | `elasticache.amazonaws.com` | The ElastiCache service principal |
| Action | `sns:Publish` | The minimum required permission |
| Resource | Topic ARN | The ARN of your specific Amazon SNS topic |
| Condition: `aws:SourceAccount` | Your AWS account ID | Restricts access to requests originating from your specific AWS account |

The condition can use either `aws:SourceAccount` or `aws:SourceOwner`. The default access policy that Amazon SNS creates for a new topic also works without changes.

### Granting permission (Console)

#### To grant ElastiCache permission to publish to your Amazon SNS topic (Console)

1. Open the Amazon Simple Notification Service console at https://console.aws.amazon.com/sns/.
2. In the navigation pane, choose **Topics**.
3. Select your topic and choose **Edit**.
4. Expand the **Access policy** section.
5. In the JSON editor, add the preceding policy statement.
6. Choose **Save changes**.

### Granting permission (AWS CLI)

Save the access policy to a file named `sns-policy.json`, then run the following command:

For Linux, macOS, or Unix:

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:region:account-id:topic-name \
    --attribute-name Policy \
    --attribute-value file://sns-policy.json \
    --region region
```

For Windows:

```
aws sns set-topic-attributes ^
    --topic-arn arn:aws:sns:region:account-id:topic-name ^
    --attribute-name Policy ^
    --attribute-value file://sns-policy.json ^
    --region region
```

To verify the policy was applied correctly, run the following command:

For Linux, macOS, or Unix:

```
aws sns get-topic-attributes \
    --topic-arn arn:aws:sns:region:account-id:topic-name \
```

```
    --query 'Attributes.Policy' \
    --output text \
    --region region
```

For Windows:

```
aws sns get-topic-attributes ^
    --topic-arn arn:aws:sns:region:account-id:topic-name ^
    --query 'Attributes.Policy' ^
    --output text ^
    --region region
```

## Adding an Amazon SNS topic

The following sections show you how to add an Amazon SNS topic using the AWS Console, the AWS CLI, or the ElastiCache API.

### Adding an Amazon SNS topic (Console)

The following procedure shows you how to add an Amazon SNS topic for a cluster. When using Valkey or Redis OSS to add an Amazon SNS topic for a replication group in step 2, instead of choosing a cluster, choose a replication group. Then follow the same remaining steps.

> **Note**
>
> This process can also be used to modify the Amazon SNS topic.

#### To add or modify an Amazon SNS topic for a cluster (Console)

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In **Clusters**, choose the cluster for which you want to add or modify an Amazon SNS topic ARN.
3. Choose **Modify**.
4. In **Modify Cluster** under **Topic for SNS Notification**, choose the SNS topic you want to add, or choose **Manual ARN input** and type the ARN of the Amazon SNS topic.
5. Choose **Modify**.

### Adding an Amazon SNS topic (AWS CLI)

To add or modify an Amazon SNS topic for a cluster, use the AWS CLI command `modify-cache-cluster`.

The following code example adds an Amazon SNS topic arn to *my-cluster*.

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-cluster \
    --cache-cluster-id my-cluster \
    --notification-topic-arn arn:aws:sns:us-
west-2:123456789xxx:ElastiCacheNotifications
```

For Windows:

```
aws elasticache modify-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --notification-topic-arn arn:aws:sns:us-west-2:123456789xx:ElastiCacheNotifications
```

For more information, see modify-cache-cluster.

### Adding an Amazon SNS topic (ElastiCache API)

To add or modify an Amazon SNS topic for a cluster, call the `ModifyCacheCluster` action with the following parameters:

- `CacheClusterId=my-cluster`
- `TopicArn=arn%3Aaws%3Asns%3Aus-west-2%3A565419523791%3AElastiCacheNotifications`

### Example

```
https://elasticache.amazon.com/
    ?Action=ModifyCacheCluster
    &ApplyImmediately=false
    &CacheClusterId=my-cluster
    &NotificationTopicArn=arn%3Aaws%3Asns%3Aus-
west-2%3A565419523791%3AElastiCacheNotifications
    &Version=2014-12-01
    &SignatureVersion=4
```

```
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

For more information, see ModifyCacheCluster.

## Enabling and disabling Amazon SNS notifications

You can turn notifications on or off for a cluster. The following procedures show you how to disable Amazon SNS notifications.

### Enabling and disabling Amazon SNS notifications (Console)

#### To disable Amazon SNS notifications using the AWS Management Console

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of your clusters running Memcached, in the navigation pane choose **Memcached**.

   To see a list of your clusters running Valkey or Redis OSS, in the navigation pane choose **Valkey** or **Redis OSS**.
3. Choose the box to the left of the cluster you want to modify notification for.
4. Choose **Modify**.
5. In **Modify Cluster** under **Topic for SNS Notification**, choose *Disable Notifications*.
6. Choose **Modify**.

### Enabling and disabling Amazon SNS notifications (AWS CLI)

To disable Amazon SNS notifications, use the command `modify-cache-cluster` with the following parameters:

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-cluster \
```

```
    --cache-cluster-id my-cluster \
    --notification-topic-status inactive
```

For Windows:

```
aws elasticache modify-cache-cluster ^
    --cache-cluster-id my-cluster ^
    --notification-topic-status inactive
```

> **Note**
>
> When the cluster belongs to a replication group, you must use the CLI command `modify-replication-group` to enable or disable SNS notifications.

### Enabling and disabling Amazon SNS notifications (ElastiCache API)

To disable Amazon SNS notifications, call the `ModifyCacheCluster` action with the following parameters:

- `CacheClusterId=my-cluster`
- `NotificationTopicStatus=inactive`

This call returns output similar to the following:

### Example

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=ModifyCacheCluster
    &ApplyImmediately=false
    &CacheClusterId=my-cluster
    &NotificationTopicStatus=inactive
    &Version=2014-12-01
    &SignatureVersion=4
    &SignatureMethod=HmacSHA256
    &Timestamp=20141201T220302Z
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Date=20141201T220302Z
    &X-Amz-SignedHeaders=Host
    &X-Amz-Expires=20141201T220302Z
```

```
    &X-Amz-Credential=<credential>
    &X-Amz-Signature=<signature>
```

## Security best practices for Amazon SNS topic policies

To help secure your Amazon SNS topic policy, follow these best practices:

- Always include the `aws:SourceAccount` condition key to prevent cross-account confused deputy attacks.
- Use the `elasticache.amazonaws.com` service principal rather than individual AWS account IDs.
- Never use `"Principal": {"AWS": "*"}` without additional condition keys.
- Grant only the minimum required permissions (`sns:Publish` only).
- Regularly audit your Amazon SNS topic access policies.

## Verifying that notifications are working

After configuring the Amazon SNS topic policy and adding the topic to your ElastiCache cluster, verify that notifications are working:

1. Trigger an ElastiCache event, such as a configuration change on your cluster.
2. Check your Amazon SNS subscription endpoint (email, Lambda function, or SQS queue) for the notification message.
3. In the Amazon SNS console, verify that the subscription status shows *Confirmed*.
4. To check the topic status, run `describe-cache-clusters` for each node and read `TopicStatus` in `NotificationConfiguration`. The `describe-replication-groups` command does not return it.

## Troubleshooting Amazon SNS notifications

If you are not receiving Amazon SNS notifications from ElastiCache, check the following:

Topic encryption

If the topic status is inactive, check whether the topic is encrypted. ElastiCache sets an encrypted topic to inactive at the first notification. Use an unencrypted topic, and then turn notifications back on.

Region mismatch

Verify that the Amazon SNS topic is in the same AWS Region as your ElastiCache cluster.

Account ownership

Verify that the Amazon SNS topic is owned by the same AWS account as your ElastiCache cluster.

Incorrect account ID in condition

Check that the `aws:SourceAccount` value is your exact 12-digit AWS account ID.

Topic ARN mismatch

Confirm that the Amazon SNS topic ARN in the policy matches exactly the ARN of the topic attached to your ElastiCache cluster.

Unconfirmed subscription

Verify that your Amazon SNS subscription is in *Confirmed* status.

Insufficient permissions

Confirm that you have `sns:SetTopicAttributes` permission to modify the topic's access policy.

# Viewing ElastiCache events

ElastiCache logs events that relate to your cluster instances, security groups, and parameter groups. This information includes the date and time of the event, the source name and source type of the event, and a description of the event. You can easily retrieve events from the log using the ElastiCache console, the AWS CLI `describe-events` command, or the ElastiCache API action `DescribeEvents`.

The following procedures show you how to view all ElastiCache events for the past 24 hours (1440 minutes).

## Viewing ElastiCache events (Console)

The following procedure displays events using the ElastiCache console.

**To view events using the ElastiCache console**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. To see a list of all available events, in the navigation pane, choose **Events**.

   On the *Events* screen each row of the list represents one event and displays the event source, the event type (cache-cluster, cache-parameter-group, cache-security-group, or cache-subnet-group), the GMT time of the event, and a description of the event.

   Using the **Filter** you can specify whether you want to see all events, or just events of a specific type in the event list.

## Viewing ElastiCache events (AWS CLI)

To generate a list of ElastiCache events using the AWS CLI, use the command `describe-events`. You can use optional parameters to control the type of events listed, the time frame of the events listed, the maximum number of events to list, and more.

The following code lists up to 40 cluster events.

```
aws elasticache describe-events --source-type cache-cluster --max-items 40
```

The following code lists all events for the past 24 hours (1440 minutes).

```
aws elasticache describe-events --source-type cache-cluster --duration 1440
```

The output from the `describe-events` command looks something like this.

```
aws elasticache describe-events --source-type cache-cluster --max-items 40
{
    "Events": [
        {
            "SourceIdentifier": "my-mem-cluster",
            "SourceType": "cache-cluster",
            "Message": "Finished modifying number of nodes from 1 to 3",
            "Date": "2020-06-09T02:01:21.772Z"
        },
        {
            "SourceIdentifier": "my-mem-cluster",
            "SourceType": "cache-cluster",
            "Message": "Added cache node 0002 in availability zone us-west-2a",
            "Date": "2020-06-09T02:01:21.716Z"
        },
        {
            "SourceIdentifier": "my-mem-cluster",
            "SourceType": "cache-cluster",
            "Message": "Added cache node 0003 in availability zone us-west-2a",
            "Date": "2020-06-09T02:01:21.706Z"
        },
        {
            "SourceIdentifier": "my-mem-cluster",
            "SourceType": "cache-cluster",
            "Message": "Increasing number of requested nodes",
            "Date": "2020-06-09T01:58:34.178Z"
        },
        {
            "SourceIdentifier": "mycluster-0003-004",
            "SourceType": "cache-cluster",
            "Message": "Added cache node 0001 in availability zone us-west-2c",
            "Date": "2020-06-09T01:51:14.120Z"
        },
        {
            "SourceIdentifier": "mycluster-0003-004",
            "SourceType": "cache-cluster",
            "Message": "This cluster does not support persistence (ex: 'appendonly').
 Please use a different instance type to enable persistence.",
            "Date": "2020-06-09T01:51:14.095Z"
```

```
        },
        {
            "SourceIdentifier": "mycluster-0003-004",
            "SourceType": "cache-cluster",
            "Message": "Cache cluster created",
            "Date": "2020-06-09T01:51:14.094Z"
        },
        {
            "SourceIdentifier": "mycluster-0001-005",
            "SourceType": "cache-cluster",
            "Message": "Added cache node 0001 in availability zone us-west-2b",
            "Date": "2020-06-09T01:42:55.603Z"
        },
        {
            "SourceIdentifier": "mycluster-0001-005",
            "SourceType": "cache-cluster",
            "Message": "This cluster does not support persistence (ex: 'appendonly').
Please use a different instance type to enable persistence.",
            "Date": "2020-06-09T01:42:55.576Z"
        },
        {
            "SourceIdentifier": "mycluster-0001-005",
            "SourceType": "cache-cluster",
            "Message": "Cache cluster created",
            "Date": "2020-06-09T01:42:55.574Z"
        },
        {
            "SourceIdentifier": "mycluster-0001-004",
            "SourceType": "cache-cluster",
            "Message": "Added cache node 0001 in availability zone us-west-2b",
            "Date": "2020-06-09T01:28:40.798Z"
        },
        {
            "SourceIdentifier": "mycluster-0001-004",
            "SourceType": "cache-cluster",
            "Message": "This cluster does not support persistence (ex: 'appendonly').
Please use a different instance type to enable persistence.",
            "Date": "2020-06-09T01:28:40.775Z"
        },
        {
            "SourceIdentifier": "mycluster-0001-004",
            "SourceType": "cache-cluster",
            "Message": "Cache cluster created",
            "Date": "2020-06-09T01:28:40.773Z"
```

```
        }
    ]
}
```

For more information, such as available parameters and permitted parameter values, see `describe-events`.

## Viewing ElastiCache events (ElastiCache API)

To generate a list of ElastiCache events using the ElastiCache API, use the `DescribeEvents` action. You can use optional parameters to control the type of events listed, the time frame of the events listed, the maximum number of events to list, and more.

The following code lists the 40 most recent cache-cluster events.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeEvents
   &MaxRecords=40
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &SourceType=cache-cluster
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The following code lists the cache-cluster events for the past 24 hours (1440 minutes).

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeEvents
   &Duration=1440
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &SourceType=cache-cluster
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

The above actions should produce output similar to the following.

```
<DescribeEventsResponse xmlns="http://elasticache.amazonaws.com/doc/2015-02-02/">
    <DescribeEventsResult>
```

```
        <Events>
            <Event>
                <Message>Cache cluster created</Message>
                <SourceType>cache-cluster</SourceType>
                <Date>2015-02-02T18:22:18.202Z</Date>
                <SourceIdentifier>mem01</SourceIdentifier>
            </Event>

 (...output omitted...)

        </Events>
    </DescribeEventsResult>
    <ResponseMetadata>
        <RequestId>e21c81b4-b9cd-11e3-8a16-7978bb24ffdf</RequestId>
    </ResponseMetadata>
</DescribeEventsResponse>
```

For more information, such as available parameters and permitted parameter values, see `DescribeEvents`.

## Event Notifications and Amazon SNS

ElastiCache can publish messages using Amazon Simple Notification Service (SNS) when significant events happen on a cluster. This feature can be used to refresh the server-lists on client machines connected to individual cache node endpoints of a cache cluster.

> **Note**
>
> For more information on Amazon Simple Notification Service (SNS), including information on pricing and links to the Amazon SNS documentation, see the Amazon SNS product page.

Notifications are published to a specified Amazon SNS *topic*. The following are requirements for notifications:

- Only one topic can be configured for ElastiCache notifications.
- The AWS account that owns the Amazon SNS topic must be the same account that owns the cluster on which notifications are enabled.
- The Amazon SNS topic must be unencrypted. ElastiCache cannot publish to a topic that is encrypted with the AWS managed key for Amazon SNS or with a customer managed key.

> **Note**
>
> You can attach an encrypted topic to a cluster, and its status shows as active. When ElastiCache sends the first notification, it sets the topic status to inactive and does not emit an event.

- The Amazon SNS topic has to be in the same Region as the ElastiCache cluster.
- The Amazon SNS topic's access policy must allow ElastiCache to publish to the topic. For the required policy configuration, see Managing ElastiCache Amazon SNS notifications.

## ElastiCache Events

The following ElastiCache events trigger Amazon SNS notifications. For information on event details, see Viewing ElastiCache events.

| Event Name | Message | Description |
|---|---|---|
| ElastiCache:AddCacheNodeComplete | `ElastiCache:AddCacheNodeComplete : cache-cluster` | A cache node has been added to the cluster and is ready for use. |
| ElastiCache:AddCacheNodeFailed due to insufficient free IP addresses | `ElastiCache:AddCacheNodeFailed : cluster-name` | A cache node could not be added because there are not enough available IP addresses . |
| ElastiCache:CacheClusterParametersChanged | `ElastiCache:CacheClusterParametersChanged : cluster-name` | One or more cluster parameters have been changed. |
| ElastiCache:CacheClusterProvisioningComplete | `ElastiCache:CacheClusterProvisioningComplete cluster-name-0001-005` | The provisioning of a cluster is completed, and the cache nodes in the cache cluster are ready to use. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CacheClusterProvisioningFailed due to incompatible network state | `ElastiCache:CacheClusterProvisioningFailed : cluster-name` | An attempt was made to launch a new cluster into a nonexistent virtual private cloud (VPC). |
| ElastiCache:CacheClusterScalingComplete | `CacheClusterScalingComplete : cluster-name` | Scaling for cache-cluster completed successfully. |
| ElastiCache:CacheClusterScalingFailed | ElastiCache:CacheClusterScalingFailed : *cluster-name* | Scale-up operation on cache-cluster failed. |
| ElastiCache:CacheClusterSecurityGroupModified | `ElastiCache:CacheClusterSecurityGroupModified : cluster-name` | One of the following events has occurred:<br>• The list of cache security groups authorized for the cluster has been modified.<br>• One or more new EC2 security groups have been authorized on any of the cache security groups associated with the cluster.<br>• One or more EC2 security groups have been revoked from any of the cache security groups associated with the cluster. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CacheNodeReplaceStarted | `ElastiCache:CacheNodeReplaceStarted : cluster-name` | ElastiCache has detected that the host running a cache node is degraded or unreachable and has started replacing the cache node.<br><br>**Note**<br>The DNS entry for the replaced cache node is not changed.<br><br>In most instances, you do not need to refresh the server-list for your clients when this event occurs. However, some cache client libraries may stop using the cache node even after ElastiCache has replaced the cache node; in this case, the application should refresh the server-list when this event occurs. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CacheNodeReplaceComplete | `ElastiCache:CacheNodeReplaceComplete : cluster-name` | ElastiCache has detected that the host running a cache node is degraded or unreachable and has completed replacing the cache node.<br><br>**Note**<br>The DNS entry for the replaced cache node is not changed.<br><br>In most instances, you do not need to refresh the server-list for your clients when this event occurs. However, some cache client libraries may stop using the cache node even after ElastiCache has replaced the cache node; in this case, the application should refresh the server-list when this event occurs. |
| ElastiCache:CacheNodesRebooted | `ElastiCache:CacheNodesRebooted : cluster-name` | One or more cache nodes has been rebooted.<br><br>Message (Memcached): `"Cache node %s shutdown"` Then a second message: `"Cache node %s restarted"` |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:CertificateRenewalComplete (Valkey or Redis OSS only) | `ElastiCache:CertificateRenewalComplete` | The Amazon CA certificate was successfully renewed. |
| ElastiCache:CreateReplicationGroupComplete | `ElastiCache:CreateReplicationGroupComplete : cluster-name` | The replication group was successfully created. |
| ElastiCache:DeleteCacheClusterComplete | `ElastiCache:DeleteCacheClusterComplete : cluster-name` | The deletion of a cluster and all associated cache nodes has completed. |
| ElastiCache:FailoverComplete (Valkey or Redis OSS only) | `ElastiCache:FailoverComplete : mycluster` | Failover over to a replica node was successful. |
| ElastiCache:ReplicationGroupIncreaseReplicaCountFinished | `ElastiCache:ReplicationGroupIncreaseReplicaCountFinished : cluster-name-0001-005` | The number of replicas in the cluster has been increased. |
| ElastiCache:ReplicationGroupIncreaseReplicaCountStarted | `ElastiCache:ReplicationGroupIncreaseReplicaCountStarted : cluster-name-0003-004` | The process of adding replicas to your cluster has begun. |
| ElastiCache:NodeReplacementCanceled | `ElastiCache:NodeReplacementCanceled : cluster-name` | A node in your cluster that was scheduled for replacement is no longer scheduled for replacement. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:NodeReplacementRescheduled | `ElastiCache:NodeReplacementRescheduled : cluster-name` | A node in your cluster previously scheduled for replacement has been rescheduled for replacement during the new window described in the notification.<br><br>For information on what actions you can take, see Replacing nodes (Valkey and Redis OSS). |
| ElastiCache:NodeReplacementScheduled | `ElastiCache:NodeReplacementScheduled : cluster-name` | A node in your cluster is scheduled for replacement during the window described in the notification.<br><br>For information on what actions you can take, see Replacing nodes (Valkey and Redis OSS). |
| ElastiCache:RemoveCacheNodeComplete | `ElastiCache:RemoveCacheNodeComplete : cluster-name` | A cache node has been removed from the cluster. |
| ElastiCache:ReplicationGroupScalingComplete | `ElastiCache:ReplicationGroupScalingComplete : cluster-name` | Scale-up operation on replication group completed successfully. |
| ElastiCache:ReplicationGroupScalingFailed | `"Failed applying modification to cache node type to %s."` | Scale-up operation on replication group failed. |

| Event Name | Message | Description |
| --- | --- | --- |
| ElastiCache:ServiceUpdateAvailableForNode | `"Service update is available for cache node %s."` | A self-service update is available for the node. |
| ElastiCache:SnapshotComplete (Valkey or Redis OSS only) | `ElastiCache:SnapshotComplete : cluster-name` | A cache snapshot has completed successfully. |
| ElastiCache:SnapshotFailed (Valkey or Redis OSS only) | `SnapshotFailed : cluster-name` | A cache snapshot has failed. See the cluster's cache events for more a detailed cause.<br><br>If you describe the snapshot, see `DescribeSnapshots` , the status will be `failed`. |

### Related topics

- Viewing ElastiCache events

# Log delivery

> **Note**
>
> Log delivery is not supported for serverless caches.
> Slow Log is supported for Valkey 7.x and above, and Redis OSS clusters and replication groups using engine version 6.0 onward.
> Engine Log is supported for Valkey 7.x and above, and Redis OSS clusters and replication groups using engine version 6.2 onward.

Log delivery lets you stream SLOWLOG or **Engine Log** to one of two destinations:

- Amazon Data Firehose
- Amazon CloudWatch Logs

You enable and configure log delivery when you create or modify a cluster using ElastiCache APIs. Each log entry will be delivered to the specified destination in one of two formats: *JSON* or *TEXT*.

A fixed number of Slow log entries are retrieved from the engine periodically. Depending on the value specified for engine parameter `slowlog-max-len`, additional slow log entries might not be delivered to the destination.

You can choose to change the delivery configurations or disable log delivery at any time using the AWS console or one of the modify APIs, either modify-cache-cluster or modify-replication-group.

You must set the `apply-immediately` parameter for all log delivery modifications.

> **Note**
>
> Amazon CloudWatch Logs charges apply when log delivery is enabled, even when logs are delivered directly to Amazon Data Firehose. For more information, see Vended Logs section in Amazon CloudWatch Pricing.

## Contents of a slow log entry

The Slow Log contains the following information:

- **CacheClusterId** – The ID of the cluster
- **CacheNodeId** – The ID of the cache node
- **Id** – A unique progressive identifier for every slow log entry
- **Timestamp** – The Unix timestamp at which the logged command was processed
- **Duration** – The amount of time needed for its execution, in microseconds
- **Command** – The command used by the client. For example, `set foo bar` where `foo` is the key and `bar` is the value. ElastiCache replaces the actual key name and value with `(2 more arguments)` to avoid exposing sensitive data.
- **ClientAddress** – Client IP address and port
- **ClientName** – Client name if set via the `CLIENT SETNAME` command

## Contents of an engine log entry

ElastiCache delivers a curated subset of engine log events, not the complete set of all internal engine logs. Only significant operational events — such as replication, snapshots, cluster health, and scripting — are included in the delivered logs.

Engine logs are designed to exclude sensitive data. Private information such as IP addresses, process IDs, file paths, and script bodies is redacted before delivery. No user data, passwords, authentication tokens, key names, or values appear in any emitted log entry.

The ElastiCache Engine Log contains the following information:

- **CacheClusterId** – The ID of the cluster
- **CacheNodeId** – The ID of the cache node
- **Log level** – LogLevel can one of the following: `VERBOSE("-")`, `NOTICE("*")`, `WARNING("#")`.
- **Time** – The UTC time of the logged message. Time is in following format: `"DD MMM YYYY hh:mm:ss.ms UTC"`
- **Role** – Role of the node from where the log is emitted. It can be one of the following: “M” for Primary, “S” for replica, "C" for writer child process working on RDB/AOF or "X" for sentinel.
- **Message** – Engine log message.

## Permissions to configure logging

You need to include the following IAM permissions in your IAM user/role policy:

- `logs:CreateLogDelivery`
- `logs:UpdateLogDelivery`
- `logs:DeleteLogDelivery`
- `logs:GetLogDelivery`
- `logs:ListLogDeliveries`

For more information, see Overview of access management: Permissions and policies.

# Log type and log format specifications

## Slow log

Slow log supports both JSON and TEXT

The following shows a JSON format example:

```
{
  "CacheClusterId": "logslowxxxxmsxj",
  "CacheNodeId": "0001",
  "Id": 296,
  "Timestamp": 1605631822,
  "Duration (us)": 0,
  "Command": "GET ... (1 more arguments)",
  "ClientAddress": "192.168.12.104:55452",
  "ClientName": "logslowxxxxmsxj##"
}
```

The following shows a TEXT format example:

```
logslowxxxxmsxj,0001,1605631822,30,GET ... (1 more
 arguments),192.168.12.104:55452,logslowxxxxmsxj##
```

## Engine log

Engine log supports both JSON and TEXT

The following shows a JSON format example:

```
{
  "CacheClusterId": "xxxxxxxxxzy-engine-log-test",
  "CacheNodeId": "0001",
  "LogLevel": "VERBOSE",
  "Role": "M",
  "Time": "12 Nov 2020 01:28:57.994 UTC",
  "Message": "Replica is waiting for next BGSAVE before synchronizing with the primary.
 Check back later"
}
```

The following shows a TEXT format example:

```
xxxxxxxxxxxzy-engine-log-test/0001:M 29 Oct 2020 20:12:20.499 UTC * A slow-running Lua
 script detected that is still in execution after 1000 microseconds.
```

# ElastiCache logging destinations

This section describes the logging destinations that you can choose for your ElastiCache logs. Each section provides guidance for configuring logging for the destination type and information about any behavior that's specific to the destination type. After you've configured your logging destination, you can provide its specifications to the ElastiCache logging configuration to start logging to it.

**Topics**


- Amazon CloudWatch Logs
- Amazon Data Firehose


## Amazon CloudWatch Logs

- You specify a CloudWatch Logs log group where the logs will be delivered.
- Logs from multiple Valkey or Redis OSS clusters and replication groups can be delivered to the same log group.
- A new log stream will be created for each node within a cluster or replication group and the logs will be delivered to the respective log streams. The log stream name will use the following format: `elasticache/${engine-name}/${cache-cluster-id}/${cache-node-id}/${log-type}`

### Permissions to publish logs to CloudWatch Logs

You must have the following permissions settings to configure ElastiCache to send logs to a CloudWatch Logs log group:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "Sid": "ElastiCacheLogging"
        },
        {
            "Sid": "ElastiCacheLoggingCWL",
            "Action": [
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

For more information, see Logs sent to CloudWatch Logs.

## Amazon Data Firehose

- You specify a Firehose delivery stream where the logs will be delivered.
- Logs from multiple Valkey or Redis OSS clusters and replication groups can be delivered to the same delivery stream.
- Logs from each node within a cluster or replication group will be delivered to the same delivery stream. You can distinguish log messages from different cache nodes based on the `cache-cluster-id` and `cache-node-id` included in each log message.
- Log delivery to Firehose is currently not available in the Asia Pacific (Osaka) Region.

### Permissions to publish logs to Firehose

You must have the following permissions to configure ElastiCache to send logs to an Amazon Kinesis Data Firehose delivery stream.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "Sid": "ElastiCacheLogging"
        },
        {
            "Sid": "ElastiCacheLoggingFHSLR",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Sid": "ElastiCacheLoggingFH",
            "Action": [
                "firehose:TagDeliveryStream"
            ],
            "Resource": "arn:aws:iam::*:role/*",
            "Effect": "Allow"
        }
    ]
}
```

# Specifying log delivery using the Console

**Log delivery requirements**

Log delivery is not supported for serverless caches.
Slow Log is supported for Valkey 7.x and above, and Redis OSS clusters and replication groups using engine version 6.0 onward.
Engine Log is supported for Valkey 7.x and above, and Redis OSS clusters and replication groups using engine version 6.2 onward.
The **Logs** tab appears in the console only when your cluster meets above requirements.

Using the AWS Management Console you can create a Valkey or Redis OSS (cluster mode disabled) cluster by following the steps at Creating a Valkey (cluster mode disabled) cluster (Console) or create a Valkey or Redis OSS (cluster mode enabled) cluster using the steps at Creating a Valkey or Redis OSS (cluster mode enabled) cluster (Console). In either case, you configure log delivery by doing the following;

1. Under **Advanced settings**, choose **Logs** and then check either **Slow logs** or **Engine logs**.
2. Under **Log format**, choose either **Text** or **JSON**.
3. Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.
4. Under **Log destination**, choose either **Create new** and enter either your Amazon S3 bucket name, CloudWatchLogs log group name or your Kinesis Data Firehose stream name, or choose **Select existing** and then choose either your CloudWatch Logs group name or your Kinesis Data Firehose stream name,

**When modifying a cluster:**

You can choose to either enable/disable log delivery or change either the destination type, format or destination:

1. Sign in to the Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Valkey clusters** or **Redis OSS clusters**.
3. From the list of clusters, choose the cluster you want to modify. Choose the **Cluster name** and not the checkbox beside it.

4. On the **Cluster name** page, choose the **Logs** tab.
5. To enable/disable slow logs, choose either **Enable slow logs** or **Disable slow logs**.
6. To enable/disable engine logs, choose either **Enable engine logs** or **Disable engine logs**.
7. To change your configuration, choose either **Modify slow logs** or **Modify engine logs**:

   - Under **Destination Type**, choose either **CloudWatch Logs** or **Kinesis Firehose**.
   - Under **Log destination**, choose either **Create new** and enter either your CloudWatchLogs log group name or your Kinesis Data Firehose stream name. Or choose **Select existing** and then choose either your CloudWatchLogs log group name or your Kinesis Data Firehose stream name.

## Specifying log delivery using the AWS CLI

### Slow Log

Create a replication group with slow log delivery to CloudWatch Logs.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id test-slow-log \
    --replication-group-description test-slow-log \
    --engine redis \
    --cache-node-type cache.r5.large \
    --num-cache-clusters 2 \
    --log-delivery-configurations '{
        "LogType":"slow-log",
        "DestinationType":"cloudwatch-logs",
        "DestinationDetails":{
          "CloudWatchLogsDetails":{
            "LogGroup":"my-log-group"
          }
        },
        "LogFormat":"json"
      }'
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id test-slow-log ^
```

```
    --replication-group-description test-slow-log ^
    --engine redis ^
    --cache-node-type cache.r5.large ^
    --num-cache-clusters 2 ^
    --log-delivery-configurations '{
        "LogType":"slow-log",
        "DestinationType":"cloudwatch-logs",
        "DestinationDetails":{
          "CloudWatchLogsDetails":{
            "LogGroup":"my-log-group"
          }
        },
        "LogFormat":"json"
      }'
```

Modify a replication group to deliver slow log to CloudWatch Logs

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id test-slow-log \
    --apply-immediately \
    --log-delivery-configurations '
    {
      "LogType":"slow-log",
      "DestinationType":"cloudwatch-logs",
      "DestinationDetails":{
        "CloudWatchLogsDetails":{

          "LogGroup":"my-log-group"
        }
      },
      "LogFormat":"json"
    }'
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id test-slow-log ^
    --apply-immediately ^
    --log-delivery-configurations '
    {
      "LogType":"slow-log",
```

```
      "DestinationType":"cloudwatch-logs",
      "DestinationDetails":{
        "CloudWatchLogsDetails":{
          "LogGroup":"my-log-group"
        }
      },
      "LogFormat":"json"
    }'
```

Modify a replication group to disable slow log delivery

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id test-slow-log \
    --apply-immediately \
    --log-delivery-configurations '
    {
      "LogType":"slow-log",
      "Enabled":false
    }'
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id test-slow-log ^
    --apply-immediately ^
    --log-delivery-configurations '
    {
      "LogType":"slow-log",
      "Enabled":false
    }'
```

**Engine Log**

Create a replication group with engine log delivery to CloudWatch Logs.

For Linux, macOS, or Unix:

```
aws elasticache create-replication-group \
    --replication-group-id test-slow-log \
    --replication-group-description test-slow-log \
```

```
    --engine redis \
    --cache-node-type cache.r5.large \
    --num-cache-clusters 2 \
    --log-delivery-configurations '{
        "LogType":"engine-log",
        "DestinationType":"cloudwatch-logs",
        "DestinationDetails":{
          "CloudWatchLogsDetails":{
            "LogGroup":"my-log-group"
          }
        },
        "LogFormat":"json"
      }'
```

For Windows:

```
aws elasticache create-replication-group ^
    --replication-group-id test-slow-log ^
    --replication-group-description test-slow-log ^
    --engine redis ^
    --cache-node-type cache.r5.large ^
    --num-cache-clusters 2 ^
    --log-delivery-configurations '{
        "LogType":"engine-log",
        "DestinationType":"cloudwatch-logs",
        "DestinationDetails":{
          "CloudWatchLogsDetails":{
            "LogGroup":"my-log-group"
          }
        },
        "LogFormat":"json"
      }'
```

Modify a replication group to deliver engine log to Firehose

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id test-slow-log \
    --apply-immediately \
    --log-delivery-configurations '
    {
      "LogType":"engine-log",
```

```
      "DestinationType":"kinesis-firehose",
      "DestinationDetails":{
      "KinesisFirehoseDetails":{
         "DeliveryStream":"test"
       }
     },
      "LogFormat":"json"
    }'
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id test-slow-log ^
    --apply-immediately ^
    --log-delivery-configurations '
    {
      "LogType":"engine-log",
      "DestinationType":"kinesis-firehose",
      "DestinationDetails":{
      "KinesisFirehoseDetails":{
         "DeliveryStream":"test"
       }
      },
      "LogFormat":"json"
    }'
```

Modify a replication group to switch to engine format

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id test-slow-log \
    --apply-immediately \
    --log-delivery-configurations '
    {
       "LogType":"engine-log",
       "LogFormat":"json"
    }'
```

For Windows:

```
aws elasticache modify-replication-group ^
```

```
    --replication-group-id test-slow-log ^
    --apply-immediately ^
    --log-delivery-configurations '
    {
       "LogType":"engine-log",
       "LogFormat":"json"
    }'
```

Modify a replication group to disable engine log delivery

For Linux, macOS, or Unix:

```
aws elasticache modify-replication-group \
    --replication-group-id test-slow-log \
    --apply-immediately \
    --log-delivery-configurations '
    {
      "LogType":"engine-log",
      "Enabled":false
    }'
```

For Windows:

```
aws elasticache modify-replication-group ^
    --replication-group-id test-slow-log ^
    --apply-immediately ^
    --log-delivery-configurations '
    {
      "LogType":"engine-log",
      "Enabled":false
    }'
```

# Monitoring use with CloudWatch Metrics

ElastiCache provides metrics that enable you to monitor your clusters. You can access these metrics through CloudWatch. For more information on CloudWatch, see the CloudWatch documentation.

ElastiCache provides both host-level metrics (for example, CPU usage) and metrics that are specific to the cache engine software (for example, cache gets and cache misses). These metrics are measured and published for each Cache node in 60-second intervals.

**Important**

You should consider setting CloudWatch alarms on certain key metrics, so that you will be notified if your cluster's performance starts to degrade. For more information, see Which Metrics Should I Monitor? in this guide.

**Topics**


- Host-Level Metrics
- Metrics for Valkey and Redis OSS
- Metrics for Memcached
- Which Metrics Should I Monitor?
- Choosing Metric Statistics and Periods
- Monitoring CloudWatch Cluster and Node Metrics


## Host-Level Metrics

The `AWS/ElastiCache` namespace includes the following host-level metrics for individual cache nodes. These metrics are measured and published for each Cache node in 60-second intervals.

**See Also**

- Metrics for Valkey and Redis OSS

| Metric | Description | Unit |
| --- | --- | --- |
| `CPUUtilization` | The percentage of CPU utilization for the entire host. Because Valkey and Redis OSS are single-threaded, we recommend you monitor `EngineCPUUtilization` metric for nodes with 4 or more vCPUs. | Percent |
| `CPUCreditBalance` | The number of earned CPU credits that an instance has accrued since it was launched or started. | Credits (vCPU-minutes) |

| Metric | Description | Unit |
| --- | --- | --- |
| | Credits are accrued in the credit balance after they are earned, and removed from the credit balance when they are spent. The credit balance has a maximum limit, determined by the instance size. After the limit is reached, any new credits that are earned are discarded.<br><br>The credits in the CPUCreditBalance are available for the instance to spend to burst beyond its baseline CPU utilization.<br><br>CPU credit metrics are available at a five-minu te frequency only. | |
| `CPUCreditUsage` | The number of CPU credits spent by the instance for CPU utilization. One CPU credit equals one vCPU running at 100% utilization for one minute or an equivalent combination of vCPUs, utilization, and time (for example, one vCPU running at 50% utilization for two minutes or two vCPUs running at 25% utilizati on for two minutes).<br><br>CPU credit metrics are available at a five-minute frequency only. If you specify a period greater than five minutes, use the Sum statistic instead of the Average statistic. | Credits (vCPU-min utes) |
| `FreeableMemory` | The amount of free memory available on the host. This is derived from the RAM, buffers, and cache that the OS reports as freeable. | Bytes |
| `NetworkBytesIn` | The number of bytes the host has read from the network. | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| NetworkBytesOut | The number of bytes sent out on all network interfaces by the instance. | Bytes |
| NetworkPacketsIn | The number of packets received on all network interfaces by the instance. This metric identifies the volume of incoming traffic in terms of the number of packets on a single instance. | Count |
| NetworkPacketsOut | The number of packets sent out on all network interfaces by the instance. This metric identifie s the volume of outgoing traffic in terms of the number of packets on a single instance. | Count |
| NetworkBandwidthIn AllowanceExceeded | The number of packets queued or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance. | Count |
| NetworkConntrackAl lowanceExceeded | The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance. | Count |
| NetworkBandwidthOu tAllowanceExceeded | The number of packets queued or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance. | Count |
| NetworkPacketsPerS econdAllowanceExce eded | The number of packets queued or dropped because the bidirectional packets per second exceeded the maximum for the instance. | Count |
| NetworkMaxBytesIn | The maximum per second burst of received bytes within each minute. | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| NetworkMaxBytesOut | The maximum per second burst of transmitted bytes within each minute. | Bytes |
| NetworkMaxPacketsIn | The maximum per second burst received packets within each minute. | Count |
| NetworkMaxPacketsOut | The maximum per second burst of transmitted packets within each minute. | Count |
| SwapUsage | The amount of swap used on the host. | Bytes |
| MajorPageFaults | The number of major page faults incurred by the engine process. High values indicate that the engine is actively swapping and experiencing memory pressure that can degrade performance. | Count |
| NetworkBaselineUsageInPercentage | The average inbound network bandwidth utilization as a percentage of the baseline bandwidth for the instance type. On instance types that support network burst (typically 4xlarge and smaller), values above 100 percent indicate that the host is consuming burst credits, a leading indicator that a sustained workload will eventually lead to credit exhaustion and throttling. | Percent |
| NetworkBaselineUsageOutPercentage | The average outbound network bandwidth utilization as a percentage of the baseline bandwidth for the instance type. On instance types that support network burst (typically 4xlarge and smaller), values above 100 percent indicate that the host is consuming burst credits, a leading indicator that a sustained workload will eventually lead to credit exhaustion and throttling. | Percent |

| Metric | Description | Unit |
| --- | --- | --- |
| NetworkBaselineMax UsageInPercentage | The maximum per-second inbound network bandwidth utilization as a percentage of the baseline bandwidth for the instance type. This metric captures microbursts that averaged metrics can hide. On instance types that support network burst (typically 4xlarge and smaller), values above 100 percent indicate that the host is consuming burst credits, a leading indicator that a sustained workload will eventually lead to credit exhaustion and throttling. | Percent |
| NetworkBaselineMax UsageOutPercentage | The maximum per-second outbound network bandwidth utilization as a percentage of the baseline bandwidth for the instance type. This metric captures microbursts that averaged metrics can hide. On instance types that support network burst (typically 4xlarge and smaller), values above 100 percent indicate that the host is consuming burst credits, a leading indicator that a sustained workload will eventually lead to credit exhaustion and throttling. | Percent |

## Metrics for Valkey and Redis OSS

The AWS/ElastiCache namespace includes the following Valkey and Redis OSS metrics. These metrics are the same when using the Valkey engine.

With the exception of ReplicationLag, EngineCPUUtilization, SuccessfulWriteRequestLatency, and SuccessfulReadRequestLatency, these metrics are derived from the **info** command. Each metric is calculated at the cache node level.

For complete documentation of the **info** command, see http://valkey.io/commands/info.

**See Also**

- Host-Level Metrics

| Metric | Description | Unit |
| --- | --- | --- |
| ActiveDefragHits | The number of value reallocations per minute performed by the active defragmentation process. This is derived from active_defrag_hits statistic at INFO. | Number |
| AuthenticationFailures | The total number of failed attempts to authenticate to Valkey or Redis OSS using the AUTH command. You can find more information about individual authentication failures using the ACL LOG command. We suggest setting an alarm on this to detect unauthorized access attempts. | Count |
| DatabaseAuthorizationFailures | The total number of failed attempts by users to access databases they don't have permission to access. This is triggered when a user's access string includes db= restrictions and the user attempts to SELECT or operate on a disallowed database. We suggest setting an alarm on this metric to detect unauthorized access attempts. Available for Valkey 9.1 and later. | Count |
| BytesUsedForCache | The total number of bytes allocated by Valkey or Redis OSS for all purposes, including the dataset, buffers, and so on. | Bytes |
| | Dimension: Tier=Memory for Valkey or Redis OSS clusters using Data tiering in ElastiCache: The total number of bytes used for cache by memory. This is the value of used_memory statistic at INFO. | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| | `Dimension: Tier=SSD` for Valkey or Redis OSS clusters using Data tiering in ElastiCache: The total number of bytes used for cache by SSD. | Bytes |
| `BytesReadFromDisk` | The total number of bytes read from disk per minute. Supported only for clusters using Data tiering in ElastiCache. | Bytes |
| `BytesWrittenToDisk` | The total number of bytes written to disk per minute. Supported only for clusters using Data tiering in ElastiCache. | Bytes |
| `CacheHits` | The number of successful read-only key lookups in the main dictionary. This is derived from `keyspace_hits` statistic at INFO. | Count |
| `CacheMisses` | The number of unsuccessful read-only key lookups in the main dictionary. This is derived from `keyspace_misses` statistic at INFO. | Count |
| `CommandAuthorizationFailures` | The total number of failed attempts by users to run commands they don't have permission to call. You can find more information about individual authentication failures using the ACL LOG command. We suggest setting an alarm on this to detect unauthorized access attempts. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CacheHitRate | Indicates the usage efficiency of the Valkey or Redis OSS instance. If the hit rate is lower than about 80 percent, it means that a significant amount of keys are evicted, expired, or don't exist. This is calculated using cache_hits and cache_misses statistics in the following way: 100 * cache_hits /(cache_hits + cache_misses) . | Percent |
| ChannelAuthorizationFailures | The total number of failed attempts by users to access channels they do not have permission to access. You can find more information about individual authentication failures using the ACL LOG command. We suggest setting an alarm on this metric to detect unauthorized access attempts. | Count |
| CurrConnections | The number of client connections, excluding connections from read replicas. ElastiCache uses 4 to 6 of the connections to monitor the cluster in each case. This is derived from the connected_clients statistic at INFO. | Count |
| CurrItems | The number of items in the cache. This is derived from the keyspace statistic, summing all of the keys in the entire keyspace. | Count |
| | Dimension: Tier=Memory for clusters using Data tiering in ElastiCache. The number of items in memory. | Count |
| | Dimension: Tier=SSD (solid state drives) for clusters using Data tiering in ElastiCache. The number of items in SSD. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CurrVolatileItems | Total number of keys in all databases that have a ttl set. This is derived from the expires statistic, summing all of the keys with a ttl set in the entire keyspace. | Count |
| CurrItemsWithVolatileFields | The number of keys that contain hash fields with expiration set. Available for Valkey 9.0 and later. | Count |
| ReclaimedFields | The number of expired hash fields reclaimed by the active expiration process. Available for Valkey 9.0 and later. | Count |
| DatabaseCapacityUsagePercentage | Percentage of the total data capacity for the cluster that is in use.<br><br>On Data Tiered instances, the metric is calculated as (used_memory - mem_not_counted_for_evict + SSD used) / (maxmemory + SSD total capacity) , where used_memory and maxmemory are taken from INFO.<br><br>In all other cases, the metric is calculated using used_memory/maxmemory . | Percent |

| Metric | Description | Unit |
| --- | --- | --- |
| DatabaseCapacityUsageCountedForEvictPercentage | Percentage of the total data capacity for the cluster that is in use, excluding the memory used for overhead and COB. This metric is calculated as:<br><br>`used_memory - mem_not_counted_for_evict/maxmemory`<br><br>On Data Tiered instances, the metric is calculated as:<br><br>`(used_memory + SSD used) / (maxmemory + SSD total capacity)`<br><br>where `used_memory` and `maxmemory` are taken from INFO | Percent |
| DatabaseMemoryUsagePercentage | Percentage of the memory for the cluster that is in use. This is calculated using `used_memory/maxmemory` from INFO. | Percent |
| DatabaseMemoryUsageCountedForEvictPercentage | Percentage of the memory for the cluster that is in use, excluding memory used for overhead and COB. This is calculated using `used_memory-mem_not_counted_for_evict/maxmemory` from INFO. | Percent |
| DB0AverageTTL | Exposes `avg_ttl` of DBO from the `keyspace` statistic of INFO command. This metric is relevant only for primary nodes. Replicas do not expire keys directly. Instead, they wait for the primary node to expire or evict keys and then receive the resulting `DEL` commands. Because replicas do not track TTL, ignore this metric on replica nodes. | Milliseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| EngineCPUUtilization | Provides CPU utilization of the Valkey or Redis OSS engine thread. Because Valkey and Redis OSS are single-threaded, you can use this metric to analyze the load of the process itself. The EngineCPUUtilization metric provides a more precise visibility of the process. You can use it in conjunction with the CPUUtilization metric. CPUUtilization exposes CPU utilization for the server instance as a whole, including other operating system and management processes. For larger node types with four vCPUs or more, use the EngineCPUUtilization metric to monitor and set thresholds for scaling.<br><br>**Note**<br>On an ElastiCache host, background processes monitor the host to provide a managed database experience. These background processes can take up a significant portion of the CPU workload. This is not significant on larger hosts with more than two vCPUs. But it can affect smaller hosts with 2vCPUs or fewer. If you only monitor the EngineCPUUtilization metric, you will be unaware of situations where the host is overloaded with both high CPU usage from Valkey or Redis OSS and high CPU usage from the background monitoring processes. Therefore, we recommend monitoring the | Percent |

| Metric | Description | Unit |
| --- | --- | --- |
| | `CPUUtilization` metric for hosts with two vCPUs or less. | |
| `Evictions` | The number of keys that have been evicted due to the `maxmemory` limit. This is derived from the `evicted_keys` statistic at INFO. | Count |
| `GlobalDatastoreReplicationLag` | This is the lag between the secondary Region's primary node and the primary Region's primary node. For cluster mode enabled Valkey or Redis OSS, the lag indicates the maximum delay among the shards. | Seconds |
| `IamAuthenticationExpirations` | The total number of expired IAM-authenticated Valkey or Redis OSS connections. You can find more information about Authenticating with IAM in ElastiCache in the user guide. | Count |
| `IamAuthenticationThrottling` | The total number of throttled IAM-authenticated Valkey or Redis OSS AUTH or HELLO requests. You can find more information about Authenticating with IAM in ElastiCache in the user guide. | Count |
| `IsMaster` | Indicates whether the node is the primary node of current shard/cluster. The metric can be either 0 (not primary) or 1 (primary). | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| KeyAuthorizationFailures | The total number of failed attempts by users to access keys they don't have permission to access. You can find more information about individual authentication failures using the ACL LOG command. We suggest setting an alarm on this to detect unauthorized access attempts. | Count |
| KeysTracked | The number of keys being tracked by Valkey or Redis OSS key tracking as a percentage of tracking-table-max-keys . Key tracking is used to aid client-side caching and notifies clients when keys are modified. | Count |
| MemoryFragmentationRatio | Indicates the efficiency in the allocation of memory of the Valkey or Redis OSS engine. Certain thresholds signify different behaviors . The recommended value is to have fragmentation above 1.0. This is calculated from the mem_fragmentation_ratio statistic of INFO. | Number |

| Metric | Description | Unit |
| --- | --- | --- |
| NewConnections | The total number of connections that have been accepted by the server during this period. This is derived from the total_connections_received statistic at INFO.<br><br>**Note**<br>If you are using ElastiCache for Redis OSS version 5 or lower, between two and four of the connections reported by this metric are used by ElastiCache to monitor the cluster. However, when using ElastiCache for Redis OSS version 6 or above, the connections used by ElastiCache to monitor the cluster are not included in this metric. | Count |
| NumItemsReadFromDisk | The total number of items retrieved from disk per minute. Supported only for clusters using Data tiering in ElastiCache. | Count |
| NumItemsWrittenToDisk | The total number of items written to disk per minute. Supported only for clusters using Data tiering in ElastiCache. | Count |
| MasterLinkHealthStatus | This status has two values: 0 or 1. The value 0 indicates that data in the ElastiCache primary node is not in sync with Valkey or Redis OSS on EC2. The value of 1 indicates that the data is in sync. To complete the migration, use the CompleteMigration API operation. | Boolean |
| Reclaimed | The total number of key expiration events. This is derived from the expired_keys statistic at INFO. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| ReplicationBytes | For nodes in a replicated configuration, ReplicationBytes reports the number of bytes that the primary is sending to all of its replicas. This metric is representative of the write load on the replication group. This is derived from the master_repl_offset statistic at INFO. | Bytes |
| ReplicationLag | This metric is only applicable for a node running as a read replica. It represents how far behind, in seconds, the replica is in applying changes from the primary node. For Valkey 7.2 and onwards, and Redis OSS 5.0.6 onwards, the lag can be measured in milliseconds. | Seconds |
| SaveInProgress | This binary metric returns 1 whenever a background save (forked or forkless) is in progress, and 0 otherwise. A background save process is typically used during snapshots and syncs. These operations can cause degraded performance. Using the SaveInProgress metric, you can diagnose whether degraded performance was caused by a background save process. This is derived from the rdb_bgsave_in_progress statistic at INFO. | Boolean |

| Metric | Description | Unit |
| --- | --- | --- |
| TrafficManagementActive | Indicates whether ElastiCache for Redis OSS is actively managing traffic by adjusting traffic allocated to incoming commands, monitoring or replication. Traffic is managed when more commands are sent to the node than can be processed by Valkey or Redis OSS and is used to maintain the stability and optimal operation of the engine. Any data points of 1 may indicate that the node is underscaled for the workload being provided.<br><br>**Note**<br>If this metric remains active, evaluate the cluster to decide if scaling up or scaling out is necessary. Related metrics include NetworkBandwidthOutAllowanceExceeded and EngineCPUUtilization. | Boolean |
| SuccessfulWriteRequestLatency | Latency of successful write requests.<br><br>Valid statistics: Average, Sum, Min, Max, Sample Count, any percentile between p0 and p100. The sample count includes only the commands that were successfully executed. | Microseconds |
| SuccessfulReadRequestLatency | Latency of successful read requests.<br><br>Valid statistics: Average, Sum, Min, Max, Sample Count, any percentile between p0 and p100. The sample count includes only the commands that were successfully executed. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| ErrorCount | The total number of failed commands during the specified time period.<br><br>Valid statistics: Average, Sum, Min, Max | Count |
| SearchNumberOfIndexes | Number of created indexes | Count |
| SearchTotalIndexedDocuments | Total number of keys in all indexes | Count |
| SearchUsedMemoryBytes | Number of bytes of memory consumed in all search data structures | Bytes |
| BlockedConnections | The number of clients currently in a blocked state, waiting on blocking commands such as BLPOP. This is derived from the blocked_clients field at INFO. | Count |
| RejectedConnections | The number of connections that the engine has rejected, primarily when the number of connected clients has reached the maxclients limit. This is derived from the rejected_connections field at INFO. | Count |
| PubSubChannels | The number of active pub/sub channels on this node that have at least one subscriber. This count does not include sharded pub/sub channels. This is derived from the pubsub_channels field at INFO. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| PubSubShardChannels | The number of active sharded pub/sub channels on this node that have at least one subscriber. Sharded channels route messages only within the shard rather than broadcasting cluster-wide. If pub/sub channel counts are growing with utilization, consider switching to sharded pub/sub to scale horizontally. This is derived from the pubsubshard_channels field at INFO. Available on Valkey 7.2 and later, and Redis OSS 7.0 and later. | Count |
| UsedMemoryDataset | An estimate of memory in bytes used by actual user data (keys and values), excluding overhead such as replication buffers, client output buffers, and internal metadata. This is derived from the used_memory_dataset field at INFO. Use this metric together with BytesUsedForCache to understand how much of your allocated memory is consumed by actual data versus overhead. Available on Valkey 7.2 and later, and Redis OSS 4.0 and later. | Bytes |
| AllocatorFragmentationBytes | The amount of memory in bytes wasted due to allocator fragmentation. Unlike MemoryFragmentationRatio, which reflects total process RSS overhead including non-actionable OS-level factors, this metric isolates fragmentation that the activedefrag engine parameter can address. This is derived from the allocator_frag_bytes field at INFO. | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| AllocatorFragmentationRatio | The ratio of allocator fragmentation relative to total allocated memory. Values above 1.0 represent fragmentation; the higher the value, the more severe the fragmentation. Unlike MemoryFragmentationRatio , which reflects total process RSS overhead including non-actionable OS-level factors, this metric isolates fragmentation that the activedefrag engine parameter can address. This is derived from the allocator_frag_ratio field at INFO. | Number |
| ProcessedCommands | The total number of commands processed by the engine. Commands rejected before execution are not counted. This is derived from the total_commands_processed field at INFO. | Count |
| DurabilityLag | The age of the oldest write acknowledged to the client but not yet persisted to the Multi-AZ transactional log. Only for durability-enabled clusters. | Milliseconds |
| DurabilityBufferExceededErrorCount | Write commands rejected due to exceeding the 10-second durability window. Only for async writes. | Count |
| MaxReplicationThroughput | The maximum observed replication throughput, sampled over short intervals and reported once per minute. Only for durability-enabled clusters. | Bytes/Second |

The following are aggregations of certain kinds of commands, derived from **info commandstats**. The commandstats section provides statistics based on the command type, including the number of calls, the total CPU time consumed by these commands, and the average CPU consumed

per command execution. For each command type, the following line is added: `cmdstat_XXX: calls=XXX,usec=XXX,usec_per_call=XXX`.

The latency metrics listed following are calculated using commandstats statistic from INFO. They are calculated in the following way: `delta(usec)/delta(calls)`. `delta` is calculated as the diff within one minute. Latency is defined as CPU time taken by ElastiCache to process the command. Note that for clusters using data tiering, the time taken to fetch items from SSD is not included in these measurements.

For a full list of available commands, see commands in the Valkey documentation.

| Metric | Description | Unit |
|---|---|---|
| `ClusterBasedCmds` | The total number of commands that are cluster-based. This is derived from the `commandstats` statistic by summing all of the commands that act upon a cluster (`cluster slot`, `cluster info`, and so on). | Count |
| `ClusterBasedCmdsLatency` | Latency of cluster-based commands. | Microseconds |
| `EvalBasedCmds` | The total number of commands for eval-based commands. This is derived from the `commandstats` statistic by summing **eval**, **evalsha**. | Count |
| `EvalBasedCmdsLatency` | Latency of eval-based commands. | Microseconds |
| `GeoSpatialBasedCmds` | The total number of commands for geospatial-based commands. This is derived from the `commandstats` statistic. It's derived by summing all of the geo type of commands: **geoadd**, **geodist**, **geohash**, **geopos**, **georadius**, and **georadiusbymember**. | Count |
| `GeoSpatialBasedCmdsLatency` | Latency of geospatial-based commands. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| GetTypeCmds | The total number of **read-only** type commands. This is derived from the commandstats statistic by summing all of the **read-only** type commands (**get**, **hget**, **scard**, **lrange**, and so on.) | Count |
| GetTypeCmdsLatency | Latency of read commands. | Microseconds |
| HashBasedCmds | The total number of commands that are hash-based. This is derived from the commandstats statistic by summing all of the commands that act upon one or more hashes (**hget**, **hkeys**, **hvals**, **hdel**, and so on). | Count |
| HashBasedCmdsLatency | Latency of hash-based commands. | Microseconds |
| HyperLogLogBasedCmds | The total number of HyperLogLog -based commands. This is derived from the commandstats statistic by summing all of the **pf** type of commands (**pfadd**, **pfcount**, **pfmerge**, and so on.). | Count |
| HyperLogLogBasedCmdsLatency | Latency of HyperLogLog-based commands. | Microseconds |
| JsonBasedCmds | The total number of JSON commands, including both read and write commands. This is derived from the commandstats statistic by summing all JSON commands that act upon JSON keys. | Count |
| JsonBasedCmdsLatency | Latency of all JSON commands, including both read and write commands. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| JsonBasedGetCmds | The total number of JSON read-only commands. This is derived from the commandstats statistic by summing all JSON read commands that act upon JSON keys. | Count |
| JsonBasedGetCmdsLatency | Latency of JSON read-only commands. | Microseconds |
| JsonBasedSetCmds | The total number of JSON write commands. This is derived from the commandstats statistic by summing all JSON write commands that act upon JSON keys. | Count |
| JsonBasedSetCmdsLatency | Latency of JSON write commands. | Microseconds |
| KeyBasedCmds | The total number of commands that are key-based. This is derived from the commandstats statistic by summing all of the commands that act upon one or more keys across multiple data structures (**del**, **expire**, **rename**, and so on.). | Count |
| KeyBasedCmdsLatency | Latency of key-based commands. | Microseconds |
| ListBasedCmds | The total number of commands that are list-based. This is derived from the commandstats statistic by summing all of the commands that act upon one or more lists (**lindex**, **lrange**, **lpush**, **ltrim**, and so on). | Count |
| ListBasedCmdsLatency | Latency of list-based commands. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| NonKeyTypeCmds | The total number of commands that are not key-based. This is derived from the `commandstats` statistic by summing all of the commands that do not act upon a key, for example, **acl**, **dbsize** or **info**. | Count |
| NonKeyTypeCmdsLatency | Latency of non-key-based commands. | Microseconds |
| `PubSubBasedCmds` | The total number of commands for pub/sub functionality. This is derived from the `commandstats` statistics by summing all of the commands used for pub/sub functionality: **psubscribe**, **publish**, **pubsub**, **punsubscribe**, **ssubscribe**, **sunsubscribe**, **spublish**, **subscribe** , and **unsubscribe**. | Count |
| `PubSubBasedCmdsLatency` | Latency of pub/sub-based commands. | Microseconds |
| `SetBasedCmds` | The total number of commands that are set-based. This is derived from the `commandstats` statistic by summing all of the commands that act upon one or more sets (**scard**, **sdiff**, **sadd**, **sunion**, and so on). | Count |
| `SetBasedCmdsLatency` | Latency of set-based commands. | Microseconds |
| `SetTypeCmds` | The total number of **write** types of commands. This is derived from the `commandstats` statistic by summing all of the **mutative** types of commands that operate on data (**set**, **hset**, **sadd**, **lpop**, and so on.) | Count |
| `SetTypeCmdsLatency` | Latency of write commands. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| SortedSetBasedCmds | The total number of commands that are sorted set-based. This is derived from the commandstats statistic by summing all of the commands that act upon one or more sorted sets (**zcount**, **zrange**, **zrank**, **zadd**, and so on). | Count |
| SortedSetBasedCmdsLatency | Latency of sorted-based commands. | Microseconds |
| StringBasedCmds | The total number of commands that are string-based. This is derived from the commandstats statistic by summing all of the commands that act upon one or more strings (**strlen**, **setex**, **setrange**, and so on). | Count |
| StringBasedCmdsLatency | Latency of string-based commands. | Microseconds |
| StreamBasedCmds | The total number of commands that are stream-based. This is derived from the commandstats statistic by summing all of the commands that act upon one or more streams data types (**xrange**, **xlen**, **xadd**, **xdel**, and so on). | Count |
| StreamBasedCmdsLatency | Latency of stream-based commands. | Microseconds |
| SearchBasedCmds | The total number of Search commands, including both read and write commands. This is derived from the commandstats statistic by summing all Search commands. | Count |
| SearchBasedCmdsLatency | Latency of all Search commands, including both read and write commands. | Microseconds |

| Metric | Description | Unit |
| --- | --- | --- |
| SearchBasedGetCmds | The total number of Search read-only commands. This is derived from the commandstats statistic by summing all Search read commands. | Count |
| SearchBasedGetCmdsLatency | Latency of Search read-only commands. | Microseconds |
| SearchBasedSetCmds | The total number of Search write commands. This is derived from the commandstats statistic by summing all Search write commands. | Count |
| SearchBasedSetCmdsLatency | Latency of Search write commands. | Microseconds |

## Metrics for Memcached

The AWS/ElastiCache namespace includes the following Memcached metrics.

The AWS/ElastiCache namespace includes the following metrics that are derived from the Memcached stats command. Each metric is calculated at the cache node level.

**See also**

- Host-Level Metrics

| Metric | Description | Unit |
| --- | --- | --- |
| BytesReadIntoMemcached | The number of bytes that have been read from the network by the cache node. | Bytes |
| BytesUsedForCacheItems | The number of bytes used to store cache items. | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| BytesWrittenOutFromMemcached | The number of bytes that have been written to the network by the cache node. | Bytes |
| CasBadval | The number of CAS (check and set) requests the cache has received where the Cas value did not match the Cas value stored. | Count |
| CasHits | The number of Cas requests the cache has received where the requested key was found and the Cas value matched. | Count |
| CasMisses | The number of Cas requests the cache has received where the key requested was not found. | Count |
| CmdFlush | The number of flush commands the cache has received. | Count |
| CmdGet | The number of get commands the cache has received. | Count |
| CmdSet | The number of set commands the cache has received. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| CurrConnections | A count of the number of connections connected to the cache at an instant in time. ElastiCache uses 2 to 3 of the connections to monitor the cluster.<br><br>In addition to the above, memcached creates a number of internal connections equal to twice the threads used for the node type. The thread count for the various node types can be see in the Nodetype Specific Parameters of the applicable Parameter Group.<br><br>The total connections is the sum of client connections, the connections for monitorin g and the internal connections mentioned above. | Count |
| CurrItems | A count of the number of items currently stored in the cache. | Count |
| DecrHits | The number of decrement requests the cache has received where the requested key was found. | Count |
| DecrMisses | The number of decrement requests the cache has received where the requested key was not found. | Count |
| DeleteHits | The number of delete requests the cache has received where the requested key was found. | Count |
| DeleteMisses | The number of delete requests the cache has received where the requested key was not found. | Count |
| Evictions | The number of non-expired items the cache evicted to allow space for new writes. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| GetHits | The number of get requests the cache has received where the key requested was found. | Count |
| GetMisses | The number of get requests the cache has received where the key requested was not found. | Count |
| IncrHits | The number of increment requests the cache has received where the key requested was found. | Count |
| IncrMisses | The number of increment requests the cache has received where the key requested was not found. | Count |
| Reclaimed | The number of expired items the cache evicted to allow space for new writes. | Count |

For Memcached 1.4.14, the following additional metrics are provided.

| Metric | Description | Unit |
| --- | --- | --- |
| BytesUsedForHash | The number of bytes currently used by hash tables. | Bytes |
| CmdConfigGet | The cumulative number of config get requests. | Count |
| CmdConfigSet | The cumulative number of config set requests. | Count |
| CmdTouch | The cumulative number of touch requests. | Count |
| CurrConfig | The current number of configurations stored. | Count |
| EvictedUnfetched | The number of valid items evicted from the least recently used cache (LRU) which were never touched after being set. | Count |

| Metric | Description | Unit |
| --- | --- | --- |
| ExpiredUnfetched | The number of expired items reclaimed from the LRU which were never touched after being set. | Count |
| SlabsMoved | The total number of slab pages that have been moved. | Count |
| TouchHits | The number of keys that have been touched and were given a new expiration time. | Count |
| TouchMisses | The number of items that have been touched, but were not found. | Count |

The AWS/ElastiCache namespace includes the following calculated cache-level metrics.

| Metric | Description | Unit |
| --- | --- | --- |
| NewConnections | The number of new connections the cache has received. This is derived from the memcached total_connections statistic by recording the change in total_connections across a period of time. This will always be at least 1, due to a connection reserved for a ElastiCache. | Count |
| NewItems | The number of new items the cache has stored. This is derived from the memcached total_items statistic by recording the change in total_items across a period of time. | Count |
| UnusedMemory | The amount of memory not used by data. This is derived from the Memcached statistics limit_maxbytes and bytes by subtracting bytes from limit_maxbytes.<br><br>Because Memcached overhead uses memory in addition to that used by data, UnusedMem | Bytes |

| Metric | Description | Unit |
| --- | --- | --- |
| | ory should not be considered to be the amount of memory available for additiona l data. You may experience evictions even though you still have some unused memory.<br><br>For more detailed information, see Memcached item memory usage. | |

# Which Metrics Should I Monitor?

The following CloudWatch metrics offer good insight into ElastiCache performance. In most cases, we recommend that you set CloudWatch alarms for these metrics so that you can take corrective action before performance issues occur.

**Metrics to Monitor**

- CPUUtilization
- EngineCPUUtilization
- SwapUsage (Valkey and Redis OSS)
- Evictions
- CurrConnections
- Memory (Valkey and Redis OSS)
- Network
- Latency
- Replication
- Traffic Management (Valkey and Redis OSS)

## CPUUtilization

This is a host-level metric reported as a percentage. For more information, see Host-Level Metrics.

**Valkey and Redis OSS**

For smaller node types with 2vCPUs or less, use the `CPUUtilization` metric to monitor your workload.

Generally speaking, we suggest you set your threshold at 90% of your available CPU. Because Valkey and Redis OSS are both single-threaded, the actual threshold value should be calculated as a fraction of the node's total capacity. For example, suppose you are using a node type that has two cores. In this case, the threshold for CPUUtilization would be 90/2, or 45%.

You will need to determine your own threshold, based on the number of cores in the cache node that you are using. If you exceed this threshold, and your main workload is from read requests, scale your cluster out by adding read replicas. If the main workload is from write requests, depending on your cluster configuration, we recommend that you:

- **Valkey or Redis OSS (cluster mode disabled) clusters:** scale up by using a larger cache instance type.
- **Valkey or Redis OSS (cluster mode enabled) clusters:** add more shards to distribute the write workload across more primary nodes.

> **Tip**
>
> Instead of using the Host-Level metric `CPUUtilization`, Valkey and Redis OSS users might be able to use the metric `EngineCPUUtilization`, which reports the percentage of usage on the Valkey or Redis OSS engine core. To see if this metric is available on your nodes and for more information, see Metrics for Valkey and Redis OSS.

For larger node types with 4vCPUs or more, you may want to use the `EngineCPUUtilization` metric, which reports the percentage of usage on the Valkey or Redis OSS engine core. To see if this metric is available on your nodes and for more information, see Metrics for Redis OSS.

**Memcached**

Because Memcached is multi-threaded, this metric can be as high as 90%. If you exceed this threshold, scale your cluster up by using a larger cache node type or scale out by adding more cache nodes.

## EngineCPUUtilization

For larger node types with 4vCPUs or more, you may want to use the `EngineCPUUtilization` metric, which reports the percentage of usage on the Redis OSS engine core. To see if this metric is available on your nodes and for more information, see Metrics for Valkey and Redis OSS.

For more information, see the **CPUs** section at Monitoring best practices with Amazon ElastiCache for Redis OSS using Amazon CloudWatch.

## SwapUsage (Valkey and Redis OSS)

This is a host-level metric reported in bytes. For more information, see Host-Level Metrics.

The `FreeableMemory` CloudWatch metric being close to 0 (i.e., below 100MB) or `SwapUsage` metric greater than the `FreeableMemory` metric indicates a node is under memory pressure. If this happens, see the following topics:

- Ensuring you have enough memory to make a Valkey or Redis OSS snapshot
- Managing reserved memory for Valkey and Redis OSS

### Evictions

This is a cache engine metric. We recommend that you determine your own alarm threshold for this metric based on your application needs.

If you are using Memcached and exceed your chosen threshold, scale your cluster up by using a larger node type or scale out by adding more nodes.

### CurrConnections

This is a cache engine metric. We recommend that you determine your own alarm threshold for this metric based on your application needs.

An increasing number of *CurrConnections* might indicate a problem with your application; you will need to investigate the application behavior to address this issue.

For more information, see the **Connections** section at Monitoring best practices with Amazon ElastiCache for Redis OSS using Amazon CloudWatch.

### Memory (Valkey and Redis OSS)

Memory is a core aspect of Valkey and Redis OSS. Understanding the memory utilization of your cluster is necessary to avoid data loss and accommodate future growth of your dataset. Statistics about the memory utilization of a node are available in the memory section of the INFO command.

For more information, see the **Memory** section at Monitoring best practices with Amazon ElastiCache for Redis OSS using Amazon CloudWatch.

### Network

One of the determining factors for the network bandwidth capacity of your cluster is the node type you have selected. For more information about the network capacity of your node, see Amazon ElastiCache pricing.

For more information, see the **Network** section at Monitoring best practices with Amazon ElastiCache for Redis OSS using Amazon CloudWatch.

## Latency

Measuring response time for an ElastiCache for Valkey instance can be approached in various ways depending on the level of granularity required. The key stages that contribute to the overall server-side response time for ElastiCache for Valkey are command pre-processing, command execution, and command post-processing.

Command-specific latency metrics derived from the Valkey INFO command such as GetTypeCmdsLatency and SetTypeCmdsLatency metric focus specifically on executing the core command logic for the Valkey command. These metrics will be helpful if your use case is to determine the command execution time or aggregated latencies per data structure.

The latency metrics `SuccessfulWriteRequestLatency` and `SuccessfulReadRequestLatency` measure the total time that the ElastiCache for Valkey engine takes to respond to a request.

> **Note**
>
> Inflated values for `SuccessfulWriteRequestLatency` and `SuccessfulReadRequestLatency` metrics can occur when using Valkey pipelining with CLIENT REPLY enabled on the Valkey client. Valkey pipelining is a technique for improving performance by issuing multiple commands at once, without waiting for the response to each individual command. To avoid inflated values, we recommend configuring your Valkey client to pipeline commands with CLIENT REPLY OFF.

For more information, see the **Latency** section at Monitoring best practices with Amazon ElastiCache using Amazon CloudWatch.

## Replication

The volume of data being replicated is visible via the `ReplicationBytes` metric. Although this metric is representative of the write load on the replication group, it doesn't provide insights into replication health. For this purpose, you can use the `ReplicationLag` metric.

For more information, see the **Replication** section at Monitoring best practices with Amazon ElastiCache for Redis OSS using Amazon CloudWatch.

## Traffic Management (Valkey and Redis OSS)

ElastiCache for Redis OSS automatically manages traffic against a node when more incoming commands are sent to the node than can be processed by Valkey or Redis OSS. This is done to maintain optimal operation and stability of the engine.

When traffic is actively managed on a node, the metric `TrafficManagementActive` will emit data points of 1. This indicates that the node may be underscaled for the workload being provided. If this metric remains 1 for long periods of time, evaluate the cluster to decide if scaling up or scaling out is necessary.

For more information, see the `TrafficManagementActive` metric on the Metrics page.

## Choosing Metric Statistics and Periods

While CloudWatch will allow you to choose any statistic and period for each metric, not all combinations will be useful. For example, the Average, Minimum, and Maximum statistics for CPUUtilization are useful, but the Sum statistic is not.

All ElastiCache samples are published for a 60 second duration for each individual cache node. For any 60 second period, a cache node metric will only contain a single sample.

For further information on how to retrieve metrics for your cache nodes, see Monitoring CloudWatch Cluster and Node Metrics.

## Monitoring CloudWatch Cluster and Node Metrics

ElastiCache and CloudWatch are integrated so you can gather a variety of metrics. You can monitor these metrics using CloudWatch.

> **Note**
>
> The following command line examples use the AWS Command Line Interface (AWS CLI). For more information, see the AWS Command Line Interface User Guide.

The following procedures show you how to use CloudWatch to gather CPU utilization statistics for a cluster.

> **Note**
>
> The `StartTime` and `EndTime` values supplied in the examples below are for illustrative purposes. You must substitute appropriate start and end time values for your cache nodes.

For information on ElastiCache limits, see AWS Service Limits for ElastiCache.

### Monitoring CloudWatch Cluster and Node Metrics (Console)

**To view metrics for the nodes in a cluster**

1. Sign in to the AWS Management Console and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. From the navigation pane, choose **Valkey caches**, **Redis OSS caches**, or **Memcached caches**, depending on the engine of your cluster.
3. Choose your cluster.
4. Choose the **Metrics** tab.

   The charts show the metrics for the nodes in the cluster. For a Valkey cluster with more than 16 nodes, the charts show the first 16 nodes by default.

   > **Note**
   >
   > The **Metrics** tab charts up to 16 nodes at a time.

5. (Valkey only, optional) To choose which nodes to chart, use the **Node ID** or **Shard ID** filter.

   If the filter matches more than 16 nodes, the console shows **Cannot select more than 16 nodes**. To see other nodes, change the filter, for example, to `Shard ID = 0003`.
6. (Redis OSS and Memcached only) To choose which nodes to chart, choose **Compare nodes**, and then select the nodes.

   For a cluster with more than 16 nodes, the charts show no metrics until you choose the nodes to chart.

(Valkey only) The **Metrics** tab has two views. The **Classic** view, which is the default, shows CloudWatch metrics. The **Detailed** view shows OpenTelemetry metrics, and appears only in Regions where detailed monitoring is available. For more information about the **Detailed** view, see Monitoring ElastiCache with OpenTelemetry metrics.

## Monitoring CloudWatch Cluster and Node Metrics using the AWS CLI

### To gather CPU utilization statistics for a cache cluster

- For Linux, macOS, or Unix:

```
aws cloudwatch get-metric-statistics \
    --namespace AWS/ElastiCache \
    --metric-name CPUUtilization \
    --dimensions='[{"Name":"CacheClusterId","Value":"test"},
{"Name":"CacheNodeId","Value":"0001"}]' \
    --statistics=Average \
```

```
    --start-time 2018-07-05T00:00:00 \
    --end-time 2018-07-06T00:00:00 \
    --period=3600
```

For Windows:

```
aws cloudwatch get-metric-statistics ^
    --namespace AWS/ElastiCache ^
    --metric-name CPUUtilization ^
    --dimensions Name=CacheClusterId,Value=test Name=CacheNodeId,Value=0001 ^
    --statistics=Average ^
    --start-time 2018-07-05T00:00:00 ^
    --end-time 2018-07-06T00:00:00 ^
    --period=3600
```

## Monitoring CloudWatch Cluster and Node Metrics using the CloudWatch API

### To gather CPU utilization statistics for a cache cluster

- Call the CloudWatch API `GetMetricStatistics` with the following parameters (note that the start and end times are shown as examples only; you will need to substitute your own appropriate start and end times):
  - `Statistics.member.1=Average`
  - `Namespace=AWS/ElastiCache`
  - `StartTime=2018-07-05T00:00:00`
  - `EndTime=2018-07-06T23:59:00`
  - `Period=3600`
  - `MetricName=CPUUtilization`
  - `Dimensions.member.1.Name=CacheClusterId`
  - `Dimensions.member.1.Value=mycachecluster`
  - `Dimensions.member.2.Name=CacheNodeId`
  - `Dimensions.member.2.Value=0002`

### Example

```
https://monitoring.amazonaws.com/
    ?Action=GetMetricStatistics
    &SignatureVersion=4
    &Version=2010-08-01
    &StartTime=2018-07-05T00:00:00
    &EndTime=2018-07-06T23:59:00
    &Period=3600
    &Statistics.member.1=Average
    &Dimensions.member.1.Name=CacheClusterId
    &Dimensions.member.1.Value=mycachecluster
    &Dimensions.member.2.Name=CacheNodeId
    &Dimensions.member.2.Value=0002
    &Namespace=AWS/ElastiCache
    &MetricName=CPUUtilization
    &Timestamp=2018-07-07T17%3A48%3A21.746Z
    &SignatureMethod=HmacSHA256
    &X-Amz-Credential=<credential>
```

# Quotas for ElastiCache

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

To view the quotas for ElastiCache, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **ElastiCache**.

To request a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*. If the quota is not yet available in Service Quotas, use the limit increase form.

Your AWS account has the following quotas related to ElastiCache.

| Name | Default value | Description | Metric Name |
|---|---|---|---|
| Nodes per Region | 300 | The maximum number of nodes across all clusters in a Region. This quota applies to your non-reserved nodes within the given Region. You can have up to 300 non-reserved nodes in the same Region. | NodesPerRegion |
| Nodes per cluster (Memcached) | 60 | The maximum number of nodes in an individual Memcached cluster. | Memcached NodesPerCluster |
| Nodes per cluster (cluster mode enabled) | 90 | The maximum number of nodes in an individual Redis OSS or Valkey cluster. | NodesPerCluster |

| Name | Default value | Description | Metric Name |
| --- | --- | --- | --- |
| Parameter groups per Region | 300 | The maximum number of parameter s groups you can create in a Region. | ParameterGroup |
| Subnet groups per Region | 300 | The maximum number of subnet groups you can create in a Region. | SubnetGroup |
| Subnets per subnet group | 20 | The maximum number of subnets you can define for a subnet group. | SubnetsPerSubnetGr oup |
| Serverless caches per Region | 40 | The maximum number of serverless caches in a Region. | ServerlessCache |
| Serverless snapshots per day per cache | 24 | The maximum number of serverles s snapshots you can take per day for each serverless cache. | SnapshotsPerDayPer Cache |
| Users per Region | 2000 | The maximum total number of users you can create in a Region. | User |
| User groups per Region | 200 | The maximum number of user groups you can create in a Region. | UserGroup |

| Name | Default value | Description | Metric Name |
| --- | --- | --- | --- |
| Users per user group | 100 | The maximum number of users you can define for a user group. | UsersPerUserGroup |
| User groups per replication group | 1 | The maximum number of user groups you can associate with a replication group. | UserGroupsPerReplicationGroup |

# Reference

The topics in this section cover working with the Amazon ElastiCache API and the ElastiCache section of the AWS CLI. Also included in this section are common error messages and service notifications.

- Using the ElastiCache API
- ElastiCache API Reference
- ElastiCache section of the AWS CLI Reference
- Amazon ElastiCache error messages
- Notifications

# Using the ElastiCache API

This section provides task-oriented descriptions of how to use and implement ElastiCache operations. For a complete description of these operations, see the Amazon ElastiCache API Reference.

**Topics**

- Using the query API
- Available libraries
- Troubleshooting applications

## Using the query API

### Query parameters

HTTP Query-based requests are HTTP requests that use the HTTP verb GET or POST and a Query parameter named `Action`.

Each Query request must include some common parameters to handle authentication and selection of an action.

Some operations take lists of parameters. These lists are specified using the `param.`*n* notation. Values of *n* are integers starting from 1.

## Query request authentication

You can only send Query requests over HTTPS and you must include a signature in every Query request. This section describes how to create the signature. The method described in the following procedure is known as *signature version 4*.

The following are the basic steps used to authenticate requests to AWS. This assumes you are registered with AWS and have an Access Key ID and Secret Access Key.

### Query authentication process

1. The sender constructs a request to AWS.
2. The sender calculates the request signature, a Keyed-Hashing for Hash-based Message Authentication Code (HMAC) with a SHA-1 hash function, as defined in the next section of this topic.
3. The sender of the request sends the request data, the signature, and Access Key ID (the key-identifier of the Secret Access Key used) to AWS.
4. AWS uses the Access Key ID to look up the Secret Access Key.
5. AWS generates a signature from the request data and the Secret Access Key using the same algorithm used to calculate the signature in the request.
6. If the signatures match, the request is considered to be authentic. If the comparison fails, the request is discarded, and AWS returns an error response.

> **Note**
>
> If a request contains a `Timestamp` parameter, the signature calculated for the request expires 15 minutes after its value.
> If a request contains an `Expires` parameter, the signature expires at the time specified by the `Expires` parameter.

### To calculate the request signature

1. Create the canonicalized query string that you need later in this procedure:

a. Sort the UTF-8 query string components by parameter name with natural byte ordering. The parameters can come from the GET URI or from the POST body (when Content-Type is application/x-www-form-urlencoded).

b. URL encode the parameter name and values according to the following rules:

   i. Do not URL encode any of the unreserved characters that RFC 3986 defines. These unreserved characters are A-Z, a-z, 0-9, hyphen ( - ), underscore ( _ ), period ( . ), and tilde ( ~ ).

   ii. Percent encode all other characters with %XY, where X and Y are hex characters 0-9 and uppercase A-F.

   iii. Percent encode extended UTF-8 characters in the form %XY%ZA....

   iv. Percent encode the space character as %20 (and not +, as common encoding schemes do).

c. Separate the encoded parameter names from their encoded values with the equals sign ( = ) (ASCII character 61), even if the parameter value is empty.

d. Separate the name-value pairs with an ampersand ( & ) (ASCII code 38).

2. Create the string to sign according to the following pseudo-grammar (the "\n" represents an ASCII newline).

```
StringToSign = HTTPVerb + "\n" +
ValueOfHostHeaderInLowercase + "\n" +
HTTPRequestURI + "\n" +
CanonicalizedQueryString <from the preceding step>
```

   The HTTPRequestURI component is the HTTP absolute path component of the URI up to, but not including, the query string. If the HTTPRequestURI is empty, use a forward slash ( / ).

3. Calculate an RFC 2104-compliant HMAC with the string you just created, your Secret Access Key as the key, and SHA256 or SHA1 as the hash algorithm.

   For more information, see https://www.ietf.org/rfc/rfc2104.txt.

4. Convert the resulting value to base64.

5. Include the value as the value of the `Signature` parameter in the request.

For example, the following is a sample request (linebreaks added for clarity).

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeCacheClusters
    &CacheClusterIdentifier=myCacheCluster
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-12-01
```

For the preceding query string, you would calculate the HMAC signature over the following string.

```
GET\n
    elasticache.amazonaws.com\n
    Action=DescribeCacheClusters
    &CacheClusterIdentifier=myCacheCluster
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-12-01
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE%2F20140523%2Fus-west-2%2Felasticache
%2Faws4_request
    &X-Amz-Date=20141201T223649Z
    &X-Amz-SignedHeaders=content-type%3Bhost%3Buser-agent%3Bx-amz-content-sha256%3Bx-
amz-date
        content-type:
        host:elasticache.us-west-2.amazonaws.com
        user-agent:CacheServicesAPICommand_Client
    x-amz-content-sha256:
    x-amz-date:
```

The result is the following signed request.

```
https://elasticache.us-west-2.amazonaws.com/
    ?Action=DescribeCacheClusters
    &CacheClusterIdentifier=myCacheCluster
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-12-01
    &X-Amz-Algorithm=&AWS;4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20141201/us-west-2/elasticache/aws4_request
    &X-Amz-Date=20141201T223649Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=2877960fced9040b41b4feaca835fd5cfeb9264f768e6a0236c9143f915ffa56
```

For detailed information on the signing process and calculating the request signature, see the topic Signature Version 4 Signing Process and its subtopics.

## Available libraries

AWS provides software development kits (SDKs) for software developers who prefer to build applications using language-specific APIs instead of the Query API. These SDKs provide basic functions (not included in the APIs), such as request authentication, request retries, and error handling so that it is easier to get started. SDKs and additional resources are available for the following programming languages:

- Java
- Windows and .NET
- PHP
- Python
- Ruby

For information about other languages, see Sample Code & Libraries.

## Troubleshooting applications

ElastiCache provides specific and descriptive errors to help you troubleshoot problems while interacting with the ElastiCache API.

### Retrieving errors

Typically, you want your application to check whether a request generated an error before you spend any time processing results. The easiest way to find out if an error occurred is to look for an `Error` node in the response from the ElastiCache API.

XPath syntax provides a simple way to search for the presence of an `Error` node, as well as an easy way to retrieve the error code and message. The following code snippet uses Perl and the XML::XPath module to determine if an error occurred during a request. If an error occurred, the code prints the first error code and message in the response.

```
use XML::XPath;
```

```
my $xp = XML::XPath->new(xml =>$response);
if ( $xp->find("//Error") )
{print "There was an error processing your request:\n", " Error code: ",
$xp->findvalue("//Error[1]/Code"), "\n", " ",
$xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

### Troubleshooting tips

We recommend the following processes to diagnose and resolve problems with the ElastiCache API.

- Verify that ElastiCache is running correctly.

  To do this, simply open a browser window and submit a query request to the ElastiCache service (such as https://elasticache.amazonaws.com). A MissingAuthenticationTokenException or 500 Internal Server Error confirms that the service is available and responding to requests.

- Check the structure of your request.

  Each ElastiCache operation has a reference page in the *ElastiCache API Reference*. Double-check that you are using parameters correctly. To give you ideas regarding what might be wrong, look at the sample requests or user scenarios to see if those examples are doing similar operations.

- Check the forum.

  ElastiCache has a discussion forum where you can search for solutions to problems others have experienced along the way. To view the forum, see

  https://forums.aws.amazon.com/ .

## Setting up the ElastiCache command line interface

This section describes the prerequisites for running the command line tools, where to get the command line tools, how to set up the tools and their environment, and includes a series of common examples of tool usage.

Follow the instructions in this topic only if you are going to the AWS CLI for ElastiCache.

**Important**

The Amazon ElastiCache Command Line Interface (CLI) does not support any ElastiCache improvements after API version 2014-09-30. To use newer ElastiCache functionality from the command line, use the AWS Command Line Interface.

**Topics**


- Prerequisites
- Getting the command line tools
- Setting up the tools
- Providing credentials for the tools
- Environmental variables


# Prerequisites

This document assumes that you can work in a Linux/UNIX or Windows environment. The Amazon ElastiCache command line tools also work on Mac OS X, which is a UNIX-based environment; however, no specific Mac OS X instructions are included in this guide.

As a convention, all command line text is prefixed with a generic **PROMPT>** command line prompt. The actual command line prompt on your machine is likely to be different. We also use **$** to indicate a Linux/UNIX specific command and **C:\>** for a Windows specific command. The example output resulting from the command is shown immediately thereafter without any prefix.

## The Java runtime environment

The command line tools used in this guide require Java version 5 or later to run. Either a JRE or JDK installation is acceptable. To view and download JREs for a range of platforms, including Linux/UNIX and Windows, see Java SE Downloads.

### Setting the Java home variable

The command line tools depend on an environment variable (`JAVA_HOME`) to locate the Java Runtime. This environment variable should be set to the full path of the directory that contains a subdirectory named `bin` which in turn contains the executable `java` (on Linux and UNIX) or `java.exe` (on Windows) executable.

**To set the Java Home variable**

1. Set the Java Home variable.

   - On Linux and UNIX, enter the following command:

     ```
     $ export JAVA_HOME=<PATH>
     ```

   - On Windows, enter the following command:

     ```
     C:\> set JAVA_HOME=<PATH>
     ```

2. Confirm the path setting by running **$JAVA_HOME/bin/java -version** and checking the output.

   - On Linux/UNIX, you will see output similar to the following:

     ```
     $ $JAVA_HOME/bin/java -version
     java version "1.6.0_23"
     Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
     Java HotSpot(TM) Client VM (build 19.0-b09, mixed mode, sharing)
     ```

   - On Windows, you will see output similar to the following:

     ```
     C:\> %JAVA_HOME%\bin\java -version
     java version "1.6.0_23"
     Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
     Java HotSpot(TM) Client VM (build 19.0-b09, mixed mode, sharing)
     ```

## Getting the command line tools

The command line tools are available as a ZIP file on the ElastiCache Developer Tools web site. These tools are written in Java, and include shell scripts for Windows 2000/XP/Vista/Windows 7, Linux/UNIX, and Mac OSX. The ZIP file is self-contained and no installation is required; simply download the zip file and unzip it to a directory on your local machine.

# Setting up the tools

The command line tools depend on an environment variable (AWS_ELASTICACHE_HOME) to locate supporting libraries. You need to set this environment variable before you can use the tools. Set it to the path of the directory you unzipped the command line tools into. This directory is named ElastiCacheCli-A.B.nnnn (A, B and n are version/release numbers), and contains subdirectories named bin and lib.

**To set the AWS_ELASTICACHE_HOME environment variable**

- Open a command line window and enter one of the following commands to set the AWS_ELASTICACHE_HOME environment variable.
  - On Linux and UNIX, enter the following command:

    ```
    $ export &AWS;_ELASTICACHE_HOME=<path-to-tools>
    ```

  - On Windows, enter the following command:

    ```
    C:\> set &AWS;_ELASTICACHE_HOME=<path-to-tools>
    ```

To make the tools easier to use, we recommend that you add the tools' BIN directory to your system PATH. The rest of this guide assumes that the BIN directory is in your system path.

**To add the tools' BIN directory to your system path**

- Enter the following commands to add the tools' BIN directory to your system PATH.
  - On Linux and UNIX, enter the following command:

    ```
    $ export PATH=$PATH:$&AWS;_ELASTICACHE_HOME/bin
    ```

  - On Windows, enter the following command:

    ```
    C:\> set PATH=%PATH%;%&AWS;_ELASTICACHE_HOME%\bin
    ```

**Note**

The Windows environment variables are reset when you close the command window. You might want to set them permanently. Consult the documentation for your version of Windows for more information.

**Note**

Paths that contain a space must be wrapped in double quotes, for example: "C:\Program Files\Java"

## Providing credentials for the tools

The command line tools need the AWS Access Key and Secret Access Key provided with your AWS account. You can get them using the command line or from a credential file located on your local system.

The deployment includes a template file ${AWS_ELASTICACHE_HOME}/credential-file-path.template that you need to edit with your information. Following are the contents of the template file:

```
AWSAccessKeyId=<Write your AWS access ID>
AWSSecretKey=<Write your AWS secret key>
```

**Important**

On UNIX, limit permissions to the owner of the credential file:

```
$ chmod 600 <the file created above>
```

With the credentials file setup, you'll need to set the AWS_CREDENTIAL_FILE environment variable so that the ElastiCache tools can find your information.

**To set the AWS_CREDENTIAL_FILE environment variable**

1. Set the environment variable:

   - On Linux and UNIX, update the variable using the following command:

     ```
     $ export &AWS;_CREDENTIAL_FILE=<the file created above>
     ```

   - On Windows, set the variable using the following command:

     ```
     C:\> set &AWS;_CREDENTIAL_FILE=<the file created above>
     ```

2. Check that your setup works properly, run the following command:

   ```
   elasticache --help
   ```

   You should see the usage page for all ElastiCache commands.

## Environmental variables

Environment variables can be useful for scripting, configuring defaults or temporarily overriding them.

In addition to the AWS_CREDENTIAL_FILE environment variable, most API tools included with the ElastiCache Command Line Interface support the following variables:

- **EC2_REGION** — The AWS region to use.
- **AWS_ELASTICACHE_URL** — The URL to use for the service call. Not required to specify a different regional endpoint if EC2_REGION is specified or the --region parameter is passed.

The following examples show how to set the environmental variable EC2_REGION to configure the region used by the API tools:

Linux, OS X, or Unix

```
$ export EC2_REGION=us-west-1
```

Windows

```
$ set EC2_REGION=us-west-1
```

# Amazon ElastiCache error messages

The following error messages are returned by Amazon ElastiCache. You may receive other error messages that are returned by ElastiCache, other AWS services, or by Valkey, Memcached, or Redis OSS. For descriptions of error messages from sources other than ElastiCache, see the documentation from the source that is generating the error message.

- Cluster node quota exceeded
- Customer's node quota exceeded
- Manual snapshot quota exceeded
- Insufficient cluster capacity
- VPC endpoint quota exceeded

Error Message: **Cluster node quota exceeded. Each cluster can have at most *%n* nodes in this region.**

> **Cause:** You attempted to create or modify a cluster with the result that the cluster would have more than *%n* nodes.
>
> **Solution:** Change your request so that the cluster does not have more than *%n* nodes. Or, if you need more than *%n* nodes, make your request using the Amazon ElastiCache Node request form.
>
> For more information, see Amazon ElastiCache Limits in *Amazon Web Services General Reference*.

Error Messages: **Customer node quota exceeded. You can have at most *%n* nodes in this region** Or, **You have already reached your quota of %s nodes in this region.**

> **Cause:** You attempted to create or modify a cluster with the result that your account would have more than *%n* nodes across all clusters in this region.

**Solution:** Change your request so that the total nodes in the region across all clusters for this account does not exceed *%n*. Or, if you need more than *%n* nodes, make your request using the Amazon ElastiCache Node request form.

For more information, see Amazon ElastiCache Limits in *Amazon Web Services General Reference*.

Error Messages: **The maximum number of manual snapshots for this cluster taken within 24 hours has been reached** or **The maximum number of manual snapshots for this node taken within 24 hours has been reached its quota of *%n***

**Cause:** You attempted to take a manual snapshot of a cluster when you have already taken the maximum number of manual snapshots allowed in a 24-hour period.

**Solution:** Wait 24 hours to attempt another manual snapshot of the cluster. Or, if you need to take a manual snapshot now, take the snapshot of another node that has the same data, such as a different node in a cluster.

Error Messages: **InsufficientCacheClusterCapacity**

**Cause:** AWS does not currently have enough available On-Demand capacity to service your request.

**Solution:**

- Wait a few minutes and then submit your request again; capacity can shift frequently.
- Submit a new request with a reduced number of nodes or shards (node groups). For example, if you're making a single request to launch 15 nodes, try making 3 requests of 5 nodes, or 15 requests for 1 node instead.
- If you're launching a cluster, submit a new request without specifying an Availability Zone.
- If you're launching a cluster, submit a new request using a different node type (which you can scale up at a later stage). For more information, see Scaling ElastiCache.

Error Messages: **Failed to create cache *cluster-name*. Quota limit reached for creating VPC endpoint.** Or, in CloudTrail: **"errorCode": "Client.VpcEndpointLimitExceeded"** and **"errorMessage": "Limit of *%n* VPC endpoints per VPC exceeded."**

**Cause:** You attempted to create an ElastiCache Serverless cluster in a VPC that has already reached its maximum number of VPC endpoints.

**Solution:** Request a quota increase for VPC endpoints:

1. Open the Service Quotas console in your AWS Region.
2. For **AWS Services**, choose **Amazon Virtual Private Cloud (Amazon VPC)**.
3. Choose **Interface VPC endpoints per VPC**, and then request an increase.

For more information, see Amazon VPC quotas in *Amazon Web Services General Reference*.

# Notifications

This topic covers ElastiCache notifications that you might be interested in. A notification is a situation or event that, in most cases, is temporary, lasting only until a solution is found and implemented. Notifications generally have a start date and a resolution date, after which the notification is no longer relevant. Any one notification might or might not be relevant to you. We recommend an implementation guideline that, if followed, improves the performance of your cluster.

Notifications do not announce new or improved ElastiCache features or functionality.

## General ElastiCache notifications

Currently there are no outstanding ElastiCache notifications that are not engine specific.

## ElastiCache for Memcached notifications

The following ElastiCache notifications are specific to the Memcached engine.

### ElastiCache for Memcached specific notifications

- Alert: Memcached LRU crawler causing segmentation faults

### Alert: Memcached LRU crawler causing segmentation faults

Alert Date: February 28, 2017
In some circumstances, your cluster might display instability with a segmentation fault in the Memcached LRU Crawler. This is an issue within the Memcached engine that has existed for some time. The issue became apparent in Memcached 1.4.33 when the LRU Crawler was enabled by default.
If you are experiencing this issue, we recommend that you disable the LRU Crawler until there is a fix. To do so, use `lru_crawler disable` at the command line or modify the `lru_crawler` parameter value (preferred).
Resolved Date:
Resolution:

## ElastiCache for Redis OSS specific notifications

There are currently no outstanding ElastiCache for Redis OSS notifications.

# ElastiCache Documentation history

- **API version:** 2015-02-02
- **Latest documentation update:** June 23, 2026

The following table describes important changes in each release of the *ElastiCache User Guide* after March 2018. For notification about updates to this documentation, you can subscribe to the RSS feed.

**Recent ElastiCache Updates**

| Change | Description | Date |
| --- | --- | --- |
| Data tiering support for R7gd and R8gd node types | ElastiCache now supports data tiering on R7gd and R8gd node types. For more information, see Data tiering. | October 28, 2026 |
| Change durability mode on existing ElastiCache clusters | You can now change the durability mode (synchronous, asynchronous, or disabled) on an existing ElastiCache for Valkey cluster without rebuilding it. For more information, see Changing durability on an existing cluster. | October 27, 2026 |
| Online migration to ElastiCache Serverless | Added Online migration to ElastiCache Serverless, which covers migrating self-managed Valkey and Redis OSS on Amazon EC2 to ElastiCache Serverless for Valkey. For more information, see Online | October 20, 2026 |

| | | |
|---|---|---|
| | migration to ElastiCache Serverless. | |
| Public endpoints for ElastiCache Serverless | Added documentation for public endpoints, a new connectivity option for ElastiCache Serverless for Valkey that provides access to caches over the internet without a VPC. For more information, see Create a Valkey serverless cache with a public endpoint. | September 29, 2026 |
| New Agent tools chapter with the ElastiCache agent skill | Added a new Agent tools for ElastiCache chapter that documents how AI coding agents can work with ElastiCache using the ElastiCache skill from the AWS Agent Toolkit. For more information, see Use the ElastiCache skill with AI coding agents. | June 29, 2026 |
| Support for ElastiCache for Memcached on-demand vertical scaling and automated horizontal scaling | ElastiCache with Memcached now supports on-demand vertical scaling and automated horizontal scaling. This includes a policy update for the action `elasticache:ModifyCacheCluster` | April 10, 2025 |

| | | |
|---|---|---|
| Support for ElastiCache with Valkey | ElastiCache now supports Valkey. Valkey 7.2.6 is compatible with Redis OSS 7.2 For more information, see Valkey. | October 8, 2024 |
| Size flexible reserved nodes | ElastiCache now supports Size flexible reserved nodes. For more information, see Amazon ElastiCache Pricing. | October 1, 2024 |
| Support for ElastiCache for Memcached 1.6.22 | ElastiCache for Memcached now supports Memcached 1.6.22. It includes cumulative bug fixes from version 1.6.18. For more information, see Memcached Version 1.6.22. | January 10, 2024 |
| ElastiCache for Redis OSS added support for additional C7gn node sizes | ElastiCache for Redis OSS added support for additional C7gn node sizes. | January 10, 2024 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS now supports creation of serverless caches | You can now create serverless caches, which simplify cache management and instantly scale to support the most demanding applications. For more information, see Choosing between deployment options. As part of this feature, new permissions were added to `ElastiCacheServiceRolePolicy` and `AmazonElastiCacheFullAccess` to allow association of serverless caches with managed VPC endpoints. Additionally, permissions were added to support a revised console experience using `AmazonElastiCacheFullAccess` policy. | November 27, 2023 |

| | | |
|---|---|---|
| ElastiCache for Memcached now supports creation of serverless caches | You can now create serverless caches, which simplify cache management and instantly scale to support the most demanding applications. For more information, see Choosing between deployment options. As part of this feature, new permissions were added to `ElastiCacheServiceRolePolicy` and `AmazonElastiCacheFullAccess` to allow association of serverless caches with managed VPC endpoints. Additionally, permissions were added to support a revised console experience using `AmazonElastiCacheFullAccess` policy. | November 27, 2023 |
| ElastiCache for Redis OSS now supports modifying cluster mode | You can now migrate clusters from Cluster Mode Disabled (CMD) to Cluster Mode Enabled (CME). For more information, see Modifying cluster mode. | May 11, 2023 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS now supports modifying in-transit encryption settings | You can now change the TLS configuration of your Redis OSS clusters without the need to re-build or re-provision the clusters or impact applicati on availability. For more information, see Enabling in-transit encryption for an existing cluster. | December 28, 2022 |
| ElastiCache for Redis OSS now supports authenticating users using IAM | IAM Authentication allows you to authenticate a connection to ElastiCache for Redis OSS using AWS IAM identities. This allows you to strengthen your security model and simplify many administrative security tasks. For more information, see Authenticating with IAM. | November 16, 2022 |
| ElastiCache for Redis OSS now supports Redis OSS 7 | This release brings several new features to Amazon ElastiCache for Redis OSS: Redis OSS functions, ACL improvements and Sharded Pub/Sub. For more informati on, see ElastiCache for Redis OSS version 7.0. | November 8, 2022 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS now supports IPV6 | ElastiCache supports the Internet Protocol versions 4 and 6 (IPv4 and IPv6), allowing you to configure your cluster to accept only IPv4 connections, only IPv6 connections or both IPv4 and IPv6 connections (dual-sta ck). IPv6 is supported for workloads using Redis OSS engine version 6.2 onward on all instances built on the Nitro system. There are no additional charges for accessing ElastiCache over IPv6. For more information, see Choosing a network type. | November 7, 2022 |
| ElastiCache for Memcached now supports IPV6 | ElastiCache supports the Internet Protocol versions 4 and 6 (IPv4 and IPv6), allowing you to configure your cluster to accept only IPv4 connections, only IPv6 connections or both IPv4 and IPv6 connections (dual-sta ck). IPv6 is supported for workloads using Memcached engine version 1.6.6 onward on all instances built on the Nitro system. There are no additional charges for accessing ElastiCache over IPv6. For more information, see Choosing a network type. | November 7, 2022 |

| | | |
|---|---|---|
| ElastiCache for Memcached now supports in-transit encryption | In-transit encryption is an optional feature that allows you to increase the security of your data at its most vulnerable points – when it is in transit from one location to another. It is supported on Memcached versions 1.6.12 and later. For more informati on, see ElastiCache in-transit encryption (TLS). | May 26, 2022 |
| ElastiCache for Redis OSS now supports native JavaScrip t Object Notation (JSON) format | The native JavaScript Object Notation (JSON) format is a simple, schemaless way to encode complex datasets inside Redis OSS clusters. You can natively store and access data using the JavaScrip t Object Notation (JSON) format inside Redis OSS clusters and update JSON data stored in those clusters, without needing to manage custom code to serialize and deserialize it. For more information, see Getting started with JSON. | May 25, 2022 |

| | | |
|---|---|---|
| ElastiCache now supports PrivateLink | AWS PrivateLink allows you to privately access ElastiCache API operations without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. For more information, see Amazon ElastiCache API and interface VPC endpoints (AWS PrivateLink) for Redis OSS or Amazon ElastiCache API and interface VPC endpoints (AWS PrivateLink) for Memcached. | January 24, 2022 |
| ElastiCache for Redis OSS now supports Redis OSS 6.2 and Data Tiering | Amazon ElastiCache for Redis OSS introduces the next version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache for Redis OSS 6.2 includes performance improvements for TLS-enabled clusters using x86 node types with 8 vCPUs or more or Graviton2 node types with 4 vCPUs or more. ElastiCache for Redis OSS also introduces data tiering. You can use data tiering as a lower-cost way to scale your clusters to up to hundreds of terabytes of capacity. For more information, see ElastiCache for Redis OSS version 6.2 (enhanced) and Data tiering. | November 23, 2021 |

| | | |
|---|---|---|
| Support for Auto Scaling | ElastiCache for Redis OSS now supports Auto Scaling. ElastiCache for Redis OSS auto scaling is the ability to increase or decrease the desired shards or replicas in your ElastiCache for Redis OSS service automatically. ElastiCache leverages the Application Auto Scaling service to provide this functionality. For more information, see Auto Scaling ElastiCache for Redis OSS clusters. | August 19, 2021 |
| Support for delivery of Redis OSS Slow logs | ElastiCache now lets you stream Redis OSS SLOWLOG to one of two destinations: Amazon Data Firehose or Amazon CloudWatch Logs. For more information, see Log delivery. | April 22, 2021 |

| | | |
|---|---|---|
| Support for tagging resources and condition keys | ElastiCache now supports tagging to help you manage your clusters and other ElastiCache resources. For more information, see Tagging your ElastiCache resources. ElastiCache also introduces support for condition keys. You can specify conditions that determine how an IAM policy takes effect. For more information, see Using condition keys. | April 7, 2021 |
| Support for tagging resources and condition keys | ElastiCache now supports tagging to help you manage your clusters and other ElastiCache resources. For more information, see Tagging your ElastiCache resources. ElastiCache also introduces support for condition keys. You can specify conditions that determine how an IAM policy takes effect. For more information, see Using condition keys. | April 7, 2021 |

| | | |
|---|---|---|
| ElastiCache is now available on AWS Outposts | AWS Outposts bring native AWS services, infrastructure, and operating models to virtually any data center, co-location space, or on-premises facility. You can deploy ElastiCache on Outposts to set up, operate, and use cache on-premises, just as you would in the cloud. For more information, see Using Outposts. | October 8, 2020 |
| ElastiCache now supports Redis OSS 6 | Amazon ElastiCache for Redis OSS introduces the next version of the Redis OSS engine supported by Amazon ElastiCache. This version includes authenticating users with role-based access control, versionless support, client-side caching, and significant operational improvements. For more information, see ElastiCache for Redis OSS Version 6.0 (Enhanced). | October 7, 2020 |

| | | |
|---|---|---|
| ElastiCache now supports Local Zones | A *Local Zone* is an extension of an AWS Region that is g eographically close to your users. You can extend any virtual private cloud (VPC) from a parent AWS Region into Local Zones by creating a new subnet and assigning it to a Local Zone. For more information, see Using Local Zones. | September 25, 2020 |
| ElastiCache for Redis OSS now supports scaling your Redis OSS Cluster environme nt up to 500 nodes or 500 shards | The Redis OSS Cluster mode makes configurations possible that you can use to partition your data across multiple shards and offers better scalability, performance, and availability. This feature is available on Amazon ElastiCac he for Redis OSS version 5.0.6 onwards in all AWS Regions and for all existing and new ElastiCache for Redis OSS Cluster environments. For more information, see Redis OSS Nodes and Shards. | August 13, 2020 |
| ElastiCache now supports resource-level permissions | You can now restrict the scope of a user's permissio ns by specifying ElastiCache resources in an AWS Identity and Access Managemen t (IAM) policy. For more information, see Resource-level permissions. | August 12, 2020 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS adds additional Amazon CloudWatch metrics | ElastiCache for Redis OSS now supports new CloudWatch metrics, including `PubSubCmds` and `HyperLogLogBasedCmds`. For a full list, see Metrics for Redis OSS. | June 10, 2020 |
| ElastiCache now supports auto-update of ElastiCache clusters | Amazon ElastiCache now supports auto-update of ElastiCache clusters after the "recommended apply by date" of service update has passed. ElastiCache will use your maintenance window to schedule the auto-update of applicable clusters. For more information, see Self-service updates. | May 13, 2020 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS now supports Global Datastore for Redis OSS | The Global Datastore for Redis OSS feature offers fully managed, fast, reliable, and secure replication across AWS Regions. Using this feature, you can create cross-Region read replica clusters for ElastiCache for Redis OSS to enable low-latency reads and disaster recovery across AWS Regions. You can create, modify, and describe a global datastore. You can also add or remove AWS Regions from your global datastore and promote an AWS Region as primary within a global datastore. For more information, see Replication Across AWS Regions Using Global Datastore. | March 16, 2020 |
| ElastiCache for Redis OSS now supports Redis OSS version 5.0.6 | For more information, see ElastiCache for Redis OSS Version 5.0.6 (Enhanced). | December 18, 2019 |

| | | |
|---|---|---|
| Amazon ElastiCache now supports T3-Standard cache nodes | You can now launch the next generation general-purpose burstable T3-Standard cache nodes in Amazon ElastiCache. Amazon EC2's T3-Standard instances provide a baseline level of CPU performance with the ability to burst CPU usage at any time until the accrued credits are exhausted . For more information, see Supported Node Types. | November 12, 2019 |
| Amazon ElastiCache now supports modifying the AUTH token on an existing ElastiCache for Redis OSS server | ElastiCache for Redis OSS 5.0.6 now enables you to modify authentication tokens by setting and rotating new tokens. You can now modify active tokens while they're in use. You can also add brand-new tokens to existing clusters enabled with encryption in transit that were previously set up without authentication tokens. This is a two-step process by which you can set and rotate the token without interrupting client requests. This feature is currently not supported on CloudFormation. For more information, see Authenticating Users with the Redis OSS AUTH Command. | October 30, 2019 |

| | | |
|---|---|---|
| Amazon ElastiCache now supports online data migration from Redis OSS on Amazon EC2 | You can now use Online Migration to migrate your data from self-hosted Redis OSS on Amazon EC2 to Amazon ElastiCache. For more information, see Online Migration to ElastiCache. | October 28, 2019 |
| ElastiCache for Redis OSS introduces online vertical scaling for Redis OSS Cluster mode. | You can now scale up or scale down your sharded Redis OSS Cluster on demand. ElastiCache for Redis OSS resizes your cluster by changing the node type, while the cluster continues to stay online and serve incoming requests. For more information, see Online Vertical Scaling by Modifying Node Type. | August 20, 2019 |
| ElastiCache for Redis OSS now allows users to use a single reader endpoint for your Amazon ElastiCache for Redis OSS cluster. | This feature allows you to direct all read traffic to your ElastiCache for Redis OSS cluster through a single, cluster-level endpoint to take advantage of load balancing and higher availability. For more information, see Finding connection endpoints. | June 13, 2019 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS now allows users to apply service updates on their own schedule | With this feature, you can choose to apply available service updates at a time of your choosing and not just during maintenance windows. This will minimize service interruptions, particularly during peak business flows, and help ensure you remain compliant if your cluster is in ElastiCache-supported compliance programs. For more information, see Self-Service Updates in Amazon ElastiCache and Compliance validation for Amazon ElastiCache. | June 4, 2019 |
| ElastiCache Standard Reserved Instance offerings : Partial Upfront, All Upfront and No Upfront. | Reserved Instances give you the flexibility to reserve an Amazon ElastiCache instance for a one- or three-year term based on an instance type and AWS Region. For more information, see Managing Costs with Reserved Nodes. | January 18, 2019 |
| ElastiCache for Redis OSS support for up to 250 nodes per Redis OSS cluster | The node or shard limit can be increased to a maximum of 250 per ElastiCache for Redis OSS cluster. For more information, see Shards. | November 19, 2018 |

| | | |
|---|---|---|
| ElastiCache for Redis OSS support for autofailover and backup and restore on all T2 nodes | ElastiCache for Redis OSS introduces support for autofailover, creating snapshots, and backup and restore on all T2 nodes. For more information, see ElastiCache for Redis OSS Backup and Restore and Snapshot. | November 19, 2018 |
| ElastiCache for Redis OSS support for M5 and R5 nodes | ElastiCache for Redis OSS now supports M5 and R5 nodes, general-purpose and memory-optimized instance types based on the AWS Nitro System. For more informati on, see Supported Node Types. | October 23, 2018 |
| Support for dynamically changing the number of read replicas | ElastiCache for Redis OSS has added support for adding and removing read replicas from any cluster with no c luster downtime. For more information about these and other changes in this release, see Changing the Number of Replicas in the *ElastiCache for Redis OSS User Guide*. See also DecreaseReplicaCount and IncreaseReplicaCount in the *ElastiCache API Reference.* | September 17, 2018 |

| | | |
|---|---|---|
| FedRAMP compliance certification | ElastiCache for Redis OSS is now certified for FedRAMP compliance. For more information, see Compliance validation for Amazon ElastiCache. | August 30, 2018 |
| Valkey or Redis OSS (cluster mode enabled) engine upgrades | Amazon ElastiCache for Redis OSS has added support for upgrading Valkey or Redis OSS (cluster mode enabled) engine versions. For more information, see Upgrading Engine Versions. | August 20, 2018 |
| PCI DSS compliance certification | ElastiCache for Redis OSS is now certified for PCI DSS compliance. For more information, see Compliance validation for Amazon ElastiCache. | July 5, 2018 |
| Support for ElastiCache for Redis OSS 4.0.10 | ElastiCache for Redis OSS now supports Redis OSS 4.0.10, including both encryption and online cluster resizing in a single version. For more information, see ElastiCache for Redis OSS Version 4.0.10 (Enhanced). | June 14, 2018 |

| | | |
|---|---|---|
| User Guide restructure | The single *ElastiCache User Guide* is now restructured so that there are separate user guides for Redis OSS (ElastiCache for Redis OSS User Guide) and for Memcached (ElastiCache for Memcached User Guide). The documentation structure in the AWS CLI Command Reference: elasticache section and the Amazon ElastiCache API Reference remain unchanged. | April 20, 2018 |
| Support for EngineCPU Utilization metric | ElastiCache for Redis OSS added a new metric, `EngineCPUUtilization` , which reports the percentage of your CPU's capacity that is currently being used. For more information, see Metrics for Redis OSS. | April 9, 2018 |

The following table describes the important changes to the *ElastiCache User Guide* before March 2018.

| Change | Description | Date Changed |
|---|---|---|
| Support for Asia Pacific (Osaka-local) Region. | ElastiCache added support for the Asia Pacific (Osaka-local) Region. The Asia Pacific (Osaka) Region currently supports a single Availability Zone and is by invitation only. For more information, see the following:<br>• Supported Regions | February 12, 2018 |

| Change | Description | Date Changed |
| --- | --- | --- |
| | • Supported cache node types | |
| Support for EU (Paris). | ElastiCache added support for the EU (Paris) Region. For more information, see the following:<br>• Supported Regions<br>• Supported cache node types | December 18, 2017 |
| Support for China (Ningxia) Region | Amazon ElastiCache added support for China (Ningxia) Region. For more information, see the following:<br>• Supported Regions<br>• Supported cache node types | December 11, 2017 |
| Support for Service Linked Roles | This release of ElastiCache added support for Service Linked Roles (SLR). For more information, see the following:<br>• Using Service-Linked Roles for Amazon ElastiCache<br>• Set up your permissions (new ElastiCache users only) | December 7, 2017 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for R4 node types | This release of ElastiCache added support R4 node types in all AWS Regions supported by ElastiCache. You can purchase R4 node types as On-Demand or as Reserved Cache Nodes. For more information, see the following:<br>• Supported cache node types<br>• Memcached node-type specific parameters<br>• Redis OSS node-type specific parameters | November 20, 2017 |
| ElastiCache for Redis OSS 3.2.10 and support for online reshardin g | Amazon ElastiCache for Redis OSS adds support for ElastiCache for Redis OSS 3.2.10. ElastiCache for Redis OSS also introduces online cluster res izing to add or remove shards from the cluster while it continues to serve incoming I/O requests. For more information, see the following:<br>• Online cluster resizing<br>• Online resharding for Valkey or Redis OSS (cluster mode enabled) | November 9, 2017 |
| HIPAA eligibility | ElastiCache for Redis OSS version 3.2.6 is now certified for HIPAA eligibility when encryption is enabled on your cluster. For more information, see the following:<br>• Compliance validation for Amazon ElastiCache<br>• Data security in Amazon ElastiCache | November 2, 2017 |

| Change | Description | Date Changed |
| --- | --- | --- |
| ElastiCache for Redis OSS 3.2.6 and support for encryption | ElastiCache adds support for ElastiCache for Redis OSS 3.2.6, which includes two encryption features:<br>• In-transit encryption encrypts your data whenever it is in transit, such as between nodes in a cluster or between a cluster and your application.<br>• At-rest encryption encrypts your on-disk data during sync and backup operations.<br><br>For more information, see the following:<br>• Data security in Amazon ElastiCache<br>• Engine versions and upgrading in ElastiCache | October 25, 2017 |
| Connection patterns topic | ElastiCache documentation adds a topic covering various patterns for accessing an ElastiCache cluster in an Amazon VPC.<br><br>For more information, see Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC in the *ElastiCache User Guide*. | April 24, 2017 |
| Support for Memcached 1.4.34 | ElastiCache adds support Memcached version 1.4.34, which incorporates a number of fixes to earlier Memcached versions.<br><br>For more information, see Memcached 1.4.34 Release Notes at Memcached on GitHub. | April 10, 2017 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for testing Automatic Failover | ElastiCache adds support for testing Automatic Failover on Redis OSS clusters that support rep lication. For more information, see the following:<br>• Testing automatic failover in the *ElastiCache User Guide*.<br>• TestFailover in the *ElastiCache API Reference*.<br>• test-failover in the *AWS CLI Reference*. | April 4, 2017 |
| Enhanced Redis OSS restore | ElastiCache adds enhanced Redis OSS backup and restore with cluster resizing. This feature supports restoring a backup to a cluster with a different number of shards than the cluster used to create the backup. (For the API and CLI, this feature can restore a different number of node groups rather than a different number of shards.) This update also supports different Redis OSS slot configurations. For more information, see Restoring from a backup into a new cache. | March 15, 2017 |
| New Redis OSS memory management parameter | ElastiCache adds a new Redis OSS parameter, `reserved-memory-percent` , which makes managing your reserved memory easier. This parameter is available on all versions of ElastiCac he for Redis OSS. For more information, see the following:<br>• Managing reserved memory for Valkey and Redis OSS<br>• New parameters for Redis OSS 3.2.4 | March 15, 2017 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for Memcached 1.4.33 | ElastiCache adds support for Memcached version 1.4.33. For more information, see the following:<br>• ElastiCache version 1.4.33 for Memcached<br>• Memcached 1.4.33 added parameters | December 20, 2016 |
| Support for EU West (London) Region | ElastiCache adds support for EU (London) Region. Only node types T2 and M4 are currently supported. For more information, see the following:<br>• Supported Regions<br>• Supported cache node types | December 13, 2016 |
| Support for Canada (Montreal) Region | ElastiCache adds support for the Canada (Montreal) Region. Only node type M4 and T2 are currently supported in this AWS Region. For more information, see the following:<br>• Supported Regions<br>• Supported cache node types | December 8, 2016 |
| Support for M4 and R3 node types | ElastiCache adds support for R3 and M4 node types in South America (São Paulo) Region and M4 node types in China (Beijing) Region. For more information, see the following:<br>• Supported Regions<br>• Supported cache node types | November 1, 2016 |
| US East 2 (Ohio) Region support | ElastiCache adds support for the US East (Ohio) Region (*us-east-2*) with M4, T2, and R3 node types. For more information, see the following:<br>• Supported Regions<br>• Supported cache node types | October 17, 2016 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for Redis OSS Cluster | ElastiCache adds support for Redis OSS Cluster (enhanced). Customers using Redis OSS Cluster, can partition their data across up to 15 shards (node groups). Each shard supports replication with up to 5 read replicas per shard. Redis OSS Cluster automatic failover times are about one fourth as long as those of earlier versions.<br><br>This release includes a redesigned managemen t console that uses terminology in keeping with industry usage.<br><br>For more information, see the following:<br><br>• Comparing Memcached and Redis OSS<br>• ElastiCache components and features — note the sections on Nodes, Shards, Clusters, and Replication.<br>• ElastiCache terminology | October 12, 2016 |
| M4 node type support | ElastiCache adds support for the M4 family of node types in most AWS Regions supported by ElastiCache. You can purchase M4 node types as On-Demand or as Reserved Cache Nodes. For more information, see the following:<br><br>• Supported cache node types<br>• Memcached node-type specific parameters<br>• Redis OSS node-type specific parameters | August 3, 2016 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Mumbai Region support | ElastiCache adds support for the Asia Pacific (Mumbai) Region. For more information, see the following:<br>• Supported cache node types<br>• Memcached node-type specific parameters<br>• Redis OSS node-type specific parameters | June 27, 2016 |
| Snapshot export | ElastiCache adds the ability to export a Redis OSS snapshot so you can access it from outside Ela stiCache. For more information, see the following:<br>• Exporting a backup in the *Amazon ElastiCache User Guide*<br>• CopySnapshot in the *Amazon ElastiCache API Reference* | May 26, 2016 |
| Node type scale up | ElastiCache adds the ability to scale up your Redis OSS node type. For more information, see Scaling ElastiCache. | March 24, 2016 |
| Easy engine upgrade | ElastiCache adds the ability to easily upgrade your Redis OSS cache engine. For more informati on, see Version Management for ElastiCache. | March 22, 2016 |
| Support for R3 node types | ElastiCache adds support for R3 node types in the China (Beijing) Region and South America (São Paulo) Region. For more information, see Supported cache node types. | March 16, 2016 |
| Accessing ElastiCache using a Lambda function | Added a tutorial on configuring a Lambda function to access ElastiCache in an Amazon VPC. For more information, see Other ElastiCache tutorials and videos. | February 12, 2016 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for Redis OSS 2.8.24 | ElastiCache adds support for Redis OSS version 2.8.24 with improvements added since Redis OSS 2.8.23. Improvements include bug fixes and support for logging bad memory access addr esses. For more information, see the following:<br>• ElastiCache version 2.8.24 for Redis OSS (enhanced)<br>• Redis OSS 2.8 Release Notes | January 20, 2016 |
| Support for Asia Pacific (Seoul) Region | ElastiCache adds support for the Asia Pacific (Seoul) (*ap-northeast-2*) Region with t2, m3, and r3 node types. | January 6, 2016 |
| Amazon ElastiCache console change. | Because the newer Redis OSS versions provide a better and more stable user experience, Redis OSS versions 2.6.13, 2.8.6, and 2.8.19 are no longer listed in the ElastiCache Management Console. For other options and more informati on, see Engine versions and upgrading in ElastiCache. | December 15, 2015 |
| Support for Redis OSS 2.8.23. | ElastiCache adds support for Redis OSS version 2.8.23 with improvements added since Redis OSS 2.8.22. Improvements include bug fixes and support for the new parameter `close-on-slave-write` which, if enabled, disconnec ts clients who attempt to write to a read-only replica. For more information, see ElastiCache version 2.8.23 for Redis OSS (enhanced). | November 13, 2015 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for Redis OSS 2.8.22. | ElastiCache adds support for Redis OSS version 2.8.22 with ElastiCache added enhanceme nts and improvements since version 2.8.21. Improvements include:<br>• Implementation of a forkless save process that enables a successful save when low available memory could cause a forked save to fail.<br>• Additional CloudWatch metrics — *SaveInPro gress* and *ReplicationBytes*.<br>• To enable partial synchronizations, the Redis OSS parameter `repl-backlog-size` now applies to all clusters.<br>For a complete list of changes and more informati on, see ElastiCache version 2.8.22 for Redis OSS (enhanced).<br>This documentation release includes a reorganiz ation of the documentation and removal of the ElastiCache command line interface (CLI) documentation. For command line use, refer to the AWS Command Line for ElastiCache. | September 28, 2015 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for Memcached 1.4.28. | ElastiCache adds support for Memcached version 1.4.24 and Memcached improvements since version 1.4.14. This release adds support for least recently used (LRU) cache management as a background task, choice of *jenkins* or *murmur3* as your hashing algorithm, new commands, and miscellaneous bug fixes. For more information, see Memcached release notes . | August 27, 2015 |
| Support for Memcached Auto Discovery using PHP 5.6 | This release of Amazon ElastiCache adds support for Memcached Auto Discovery client for PHP version 5.6. For more information, see Compiling the source code for the ElastiCache cluster client for PHP. | July 29, 2015 |
| Support for Redis OSS 2.8.21 | ElastiCache adds support for Redis OSS version 2.8.21 and Redis OSS improvements since version 2.8.19. This Redis OSS release includes several bug fixes. For more information, see Redis OSS 2.8 release notes. | July 29, 2015 |
| New topic: Accessing ElastiCache from outside AWS | Added new topic on how to access ElastiCache resources from outside AWS. For more informati on, see Accessing ElastiCache resources from outside AWS. | July 9, 2015 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Node replacement messages added | ElastiCache adds three messages pertaining to scheduled node replacement, ElastiCache:NodeReplacementScheduled, ElastiCache:NodeReplacementRescheduled, and ElastiCache:NodeReplacementCanceled.<br><br>For more information and actions you can take when a node is scheduled for replacement, see ElastiCache's Event Notifications and Amazon SNS. | June 11, 2015 |
| Support for Redis OSS v. 2.8.19. | ElastiCache adds support for Redis OSS version 2.8.19 and Redis OSS improvements since version 2.8.6. This support includes support for:<br><br>• The HyperLogLog data structure, with the Redis OSS commands PFADD, PFCOUNT, and PFMERGE.<br>• Lexicographic range queries with the new commands ZRANGEBYLEX, ZLEXCOUNT, and ZREMRANGEBYLEX.<br>• Introduced a number of bug fixes, namely preventing a primary node from sending stale data to replica nodes by failing the primary SYNC when a background save (bgsave) child process terminates unexpectedly.<br><br>For more information on HyperLogLog, see Redis OSS new data structure: the HyperLogLog.<br><br>For more information on PFADD, PFCOUNT, and PFMERGE, see the Redis OSS Documentation and click **HyperLogLog**. | March 11, 2015 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for cost allocation tags | ElastiCache adds support for cost allocation tags. For more information, see Monitoring costs with cost allocation tags. | February 9, 2015 |
| Support for AWS GovCloud (US-West) Region | ElastiCache adds support for the AWS GovCloud (US-West) (*us-gov-west-1*) Region. | January 29, 2015 |
| Support for Europe (Frankfurt) Region | ElastiCache adds support for the Europe (Frankfurt) (*eu-central-1*) Region. | January 19, 2015 |
| Multi-AZ support for Redis OSS replication groups | ElastiCache adds support for Multi-AZ from the primary node to a read replica in a Redis OSS replication group. ElastiCache monitors the health of the replication group. If the primary fails, ElastiCache automatically promotes a replica to primary, then replaces the replica. For more information, see Minimizing downtime in ElastiCache by using Multi-AZ with Valkey and Redis OSS. | October 24, 2014 |
| AWS CloudTrail logging of API calls supported | ElastiCache adds support for using AWS CloudTrail to log all ElastiCache API calls. For more information, see Logging Amazon ElastiCache API calls with AWS CloudTrail. | September 15, 2014 |
| New instance sizes supported | ElastiCache adds support for additional General Purpose (T2) instances. For more information, see Configuring engine parameters using ElastiCache parameter groups. | September 11, 2014 |
| Flexible node placement supported for Memcached | ElastiCache adds support for creating Memcached nodes across multiple Availability Zones. | July 23, 2014 |

| Change | Description | Date Changed |
| --- | --- | --- |
| New instance sizes supported | ElastiCache adds support for additional General Purpose (M3) instances and Memory Optimized (R3) instances. For more information, see Configuring engine parameters using ElastiCache parameter groups. | July 1, 2014 |
| PHP auto discovery | Added support for PHP version 5.5 auto discovery . | May 13, 2014 |
| Backup and restore for Redis OSS clusters | In this release, ElastiCache allows customers to create snapshots of their Redis OSS clusters, and create new clusters using these snapshots . A backup is a copy of the cluster at a specific moment in time, and consists of cluster metadata and all of the data in the Redis OSS cache. Backups are stored in Amazon S3, and customers can restore the data from a snapshot into a new cluster at any time. For more informati on, see Snapshot and restore. | April 24, 2014 |
| Redis OSS 2.8.6 | ElastiCache supports Redis OSS 2.8.6, in addition to Redis OSS 2.6.13. With Redis OSS 2.8.6, customers can improve the resiliency and fault tolerance of read replicas, with support for partial resynchronization, and a user-defined minimum number of read replicas that must be available at all times. Redis OSS 2.8.6 also offers full support for publish-and-subscribe, where clients can be notified of events that occur on the server. | March 13, 2014 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Redis OSS cache engine | ElastiCache offers Redis OSS cache engine software, in addition to Memcached. Customers who currently use Redis OSS can "seed" a new ElastiCache Redis OSS cluster with their existing data from a Redis OSS snapshot file, easing migration to a managed ElastiCache environment.<br><br>To support Redis OSS replication capabilities, the ElastiCache API now supports replication groups. Customers can create a replication group with a primary Redis OSS cache node, and add one or more read replica nodes that automa tically stay synchronized with cache data in the primary node. Read-intensive applications can be offloaded to a read replica, reducing the load on the primary node. Read replicas can also guard against data loss in the event of a primary cache node failure. | September 3, 2013 |
| Support for default Amazon Virtual Private Cloud (VPC) | In this release, ElastiCache is fully integrated with Amazon Virtual Private Cloud (VPC). For new customers, clusters are created in an Amazon VPC by default. For more information, see Amazon VPCs and ElastiCache security. | January 8, 2013 |
| PHP support for cache node auto discovery | The initial release of cache node auto discovery provided support for Java programs. In this release, ElastiCache brings cache node auto discovery support to PHP. | January 2, 2013 |

| Change | Description | Date Changed |
| --- | --- | --- |
| Support for Amazon Virtual Private Cloud (VPC) | In this release, ElastiCache clusters can be launched in Amazon Virtual Private Cloud (VPC). By default, new customers' clusters are created in an Amazon VPC automatically; existing customers can migrate to Amazon VPC at their own pace. For more information, see Amazon VPCs and ElastiCache security. | December 20, 2012 |
| Cache node auto discovery and new cache engine version | ElastiCache provides cache node auto discovery—the ability for client programs to automatically determine all of the cache nodes in a cluster, a nd to initiate and maintain connections to all of these nodes.<br><br>This release also offers a new cache engine version: Memcached version 1.4.14. This new cache engine provides enhanced slab rebalanci ng capability, significant performance and scalability improvements, and several bug fixes. There are several new cache parameters that can be configured. For more information, see Configuring engine parameters using ElastiCache parameter groups. | November 28, 2012 |
| New cache node types | This release provides four additional cache node types. | November 13, 2012 |
| Reserved cache nodes | This release adds support for reserved cache nodes.<br>q | April 5, 2012 |
| New guide | This is the first release of *Amazon ElastiCache User Guide*. | August 22, 2011 |

# AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.