User Guide

Amazon ElastiCache (Redis OSS)

API Version 2015-02-02

Amazon ElastiCache (Redis OSS): User Guide

Table of Contents

What is Amazon ElastiCache (Redis OSS)?

Welcome to the *Amazon ElastiCache (Redis OSS) 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 choose to design your own cache cluster.

> **Note**
>
> Amazon ElastiCache works with both the Redis OSS and Memcached engines. Use the guide for the engine that you're interested in. If you're unsure which engine you want to use, see Comparing Memcached and Redis OSS self-designed caches in this guide.

Serverless caching

ElastiCache (Redis OSS) offers serverless caching, which simplifies adding and operating a Redis-based cache for your application. ElastiCache (Redis OSS) 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 (Redis OSS) Serverless also removes the need to plan and manage caching capacity. ElastiCache (Redis OSS) constantly monitors the cache’s memory, compute, and network bandwidth used by your application, and scales to meet the needs of your application. ElastiCache (Redis OSS) offers a simple endpoint experience for developers, by abstracting the underlying cache infrastructure and cluster design. ElastiCache (Redis OSS) 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 (Redis OSS) Serverless is compatible with Redis OSS 7.1 and above.

Designing your own ElastiCache (Redis OSS) cluster

If you need fine-grained control over your ElastiCache (Redis OSS) cluster, you can choose to design your own Redis OSS cluster with ElastiCache. ElastiCache enables you to design your 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.

Designing your own ElastiCache (Redis OSS) 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 Redis OSS in cluster mode enabling horizontal scaling, or without cluster mode for just scaling vertically. When designing your own clusters, 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 Redis OSS cluster.

When designing your own ElastiCache (Redis OSS) cluster, you can choose to run Redis OSS 3.0 and above.

Related services

MemoryDB

When deciding whether to use ElastiCache (Redis OSS) or MemoryDB consider the following comparisons:

- ElastiCache (Redis OSS) is a service that is commonly used to cache data from other databases and data stores using Redis OSS. You should consider ElastiCache (Redis OSS) 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 (Redis OSS) for use cases where you want to use the Redis OSS data structures and APIs to access data stored in a primary database or data store.
- MemoryDB is a durable, in-memory database for workloads that require an ultra-fast, primary database. You should consider using MemoryDB if your workload requires a durable database that provides ultra-fast performance (microsecond read and single-digit millisecond write latency). MemoryDB may also be a good fit for your use case if you want to build an application using Redis OSS data structures and APIs with a primary, durable database. Finally, you should

consider using MemoryDB to simplify your application architecture and lower costs by replacing usage of a database with a cache for durability and performance.

Amazon RDS

ElastiCache (Redis OSS) can 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.

How it works

Here you can find an overview of the major components of an ElastiCache (Redis OSS) 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 (Redis OSS) offers two deployment options: Serverless and self-designed clusters. See Choosing between deployment options

> **Note**
>
> Amazon ElastiCache works with both the Redis OSS and Memcached engines. Use the guide for the engine that you're interested in. If you're unsure which engine you want to use, see Comparing Memcached and Redis OSS self-designed caches in this guide.

Topics

How ElastiCache (Redis OSS) works

ElastiCache (Redis OSS) Serverless

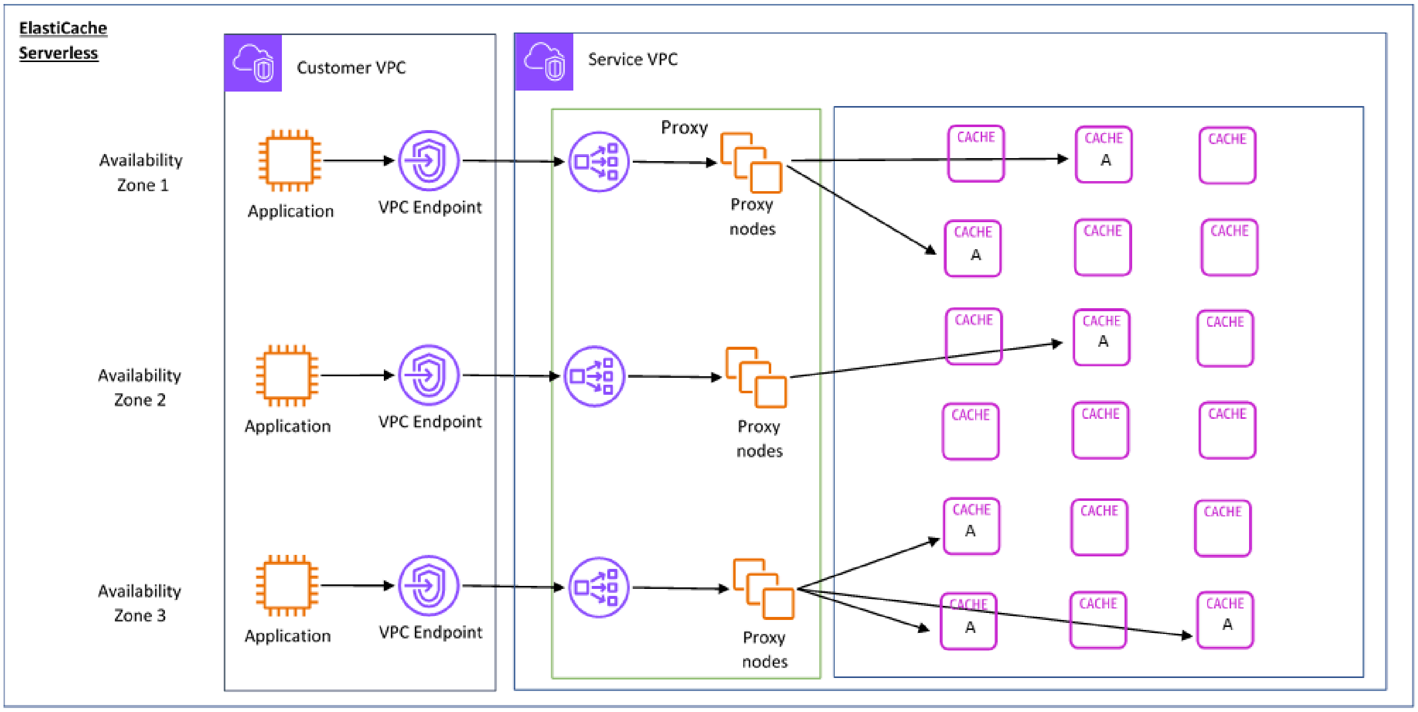
ElastiCache (Redis OSS) Serverless enables you to create a cache without worrying about capacity planning, hardware management, or cluster design. You simply provide a name for your cache and you receive a single endpoint that you can configure in your Redis OSS client to begin accessing your cache.

> **Note**
>
> ElastiCache (Redis OSS) Serverless runs Redis OSS in cluster mode and is only compatible with Redis OSS clients that support both TLS and the Redis OSS cluster protocol.

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 Redis OSS 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, ElastiCache creates a Virtual Private Cloud (VPC) Endpoint in the subnets of your choice in your VPC. Your application can connect to the cache through these VPC Endpoints.

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 cache 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.

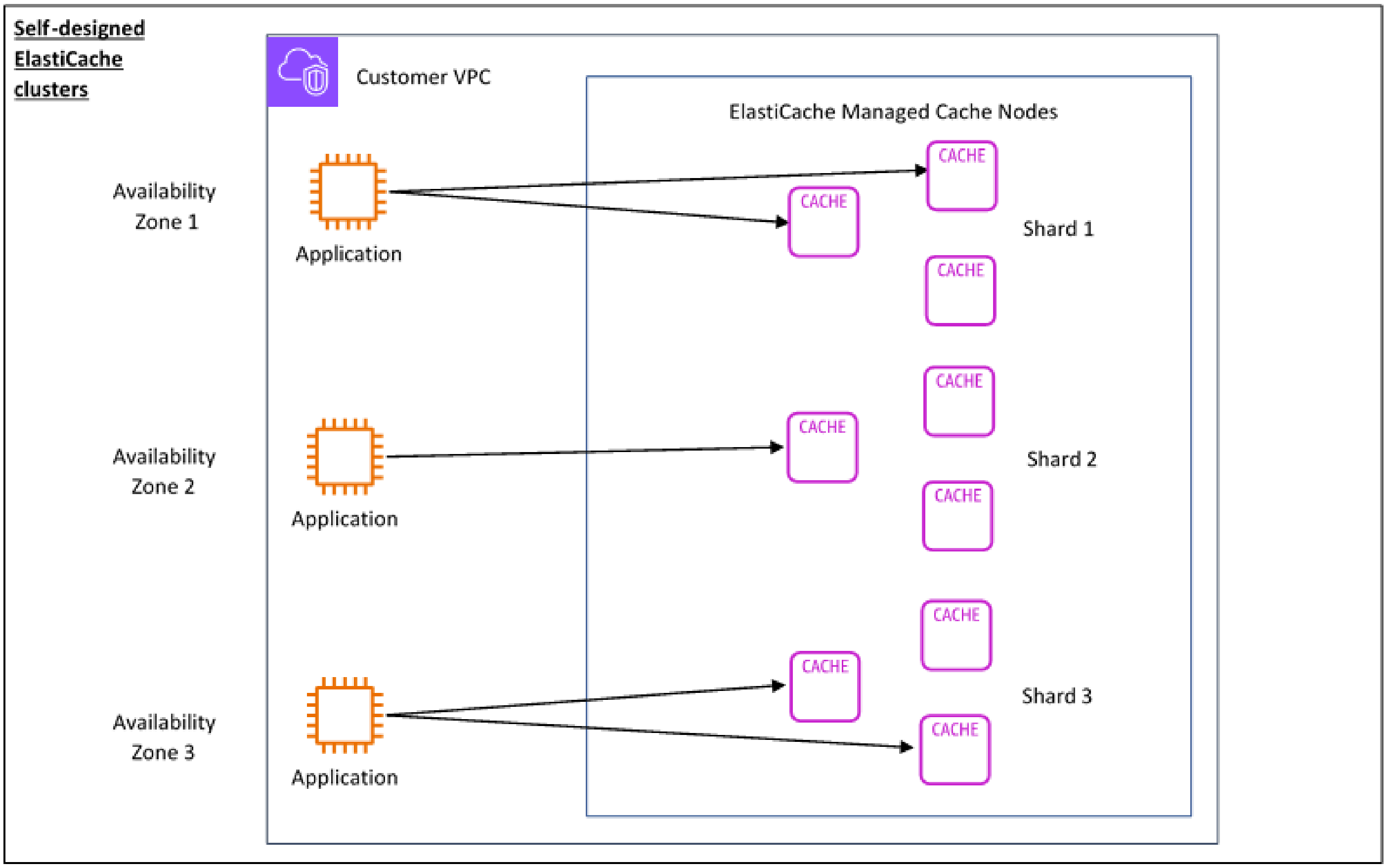
Self-designed ElastiCache clusters

You can choose to design your own ElastiCache clusters by choosing an cache node family, size, and number of nodes for your cluster. Designing your own 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 Redis OSS in cluster mode by creating a cluster with multiple shards, or in non-cluster mode with a single shard.

Key Benefits

- **Design your own cluster:** With ElastiCache, you can design your own 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 design your cluster with nodes across multiple AZs to achieve high-availability.
- **Fine-grained control:** When designing our own cluster, you have more control over fine-tuning the settings on your cache. For example, you can use Redis OSS-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 ElastiCache self-designed clusters work.



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 choosing to design your own ElastiCache (Redis OSS) clusters, 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 Redis OSS 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.
- Commands that require additional vCPU time will consume proportionally more ECPUs. For example, if your application uses the Redis OSS HMGET command, and consumes 3 times the vCPU time as a simple SET/GET command, then it will consume 3 ECPUs.
- 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.

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

Node hours

You can choose to design your own Redis OSS cache cluster by choosing the EC2 node family, size, number of nodes, and placement across Availability Zones. When self-designing your cluster, you pay per hour for each cache node.

ElastiCache for Redis OSS backups

A *backup* is a point-in-time copy of a Redis OSS cache. 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
- Self-designed clusters

For a list of supported commands for both, see Supported and restricted 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 cache cluster capacity. ElastiCache Serverless automatically stores data redundantly across three Availability Zones and provides a 99.99% availability Service Level Agreement (SLA). Backups are cross-compatible, and can be exported to and restored from Self-designed clusters.

Self-designed clusters

If you need fine-grained control over your ElastiCache (Redis OSS) cluster, you can choose to design your own Redis OSS cluster with ElastiCache. 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 helps manage hardware provisioning, monitoring, node replacements, and software patching for your cluster. Self-designed clusters can be designed to provide an up to 99.99% availibility SLA. Backups are cross-compatible, and can be exported to and restored from Serveless caches.

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.

Choose to design your own ElastiCache cluster if:

- You are already running ElastiCache Serverless and want finer grained control over the type of node running 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.

Comparing serverless caching and self-designed clusters

Feature	Serverless caching	Self-designed clusters
Cache setup	Create a cache with just a name in under a minute	Provides fine-grained control over cache cluster design. User can choose node-type, number of nodes, and placement across AWS availability zones
Supported ElastiCache (Redis OSS) version	ElastiCache (Redis OSS) version 7.1 and higher	ElastiCache (Redis OSS) version 4.0 and higher
Cluster Mode	Operates Redis OSS in `cluster mode enabled` only. Redis OSS 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 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. 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. With the Auto-Scaling feature you can also configure scaling based on a schedule, or scale

Feature	Serverless caching	Self-designed clusters
		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, replaceme nts, 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 connectio ns.
Configurability	No fine-grained configura tion 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.	Self-designed 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 [Redis OSS node-type specific para meters]().
Multi-AZ	Data is replicated asynchron ously across multiple Availabil ity Zones for higher availabil ity and improved read latency.	Provides an option to design the cluster in a single Availability Zone or across multiple Availability Zones (AZs). For Multi-AZ clusters, data is replicated asynchron ously across multiple Availabil ity Zones for higher availabil ity and improved read latency.

Feature	Serverless caching	Self-designed clusters
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.
Backups	Supports automatic and manual backups of caches with no performance impact. Backups are cross-compatible, and can be restored into an ElastiCache Serverless cache or a self-designed cluster.	Supports automatic and manual backups. Clusters may see some performance impact depending on the available reserved memory. For more information, see Managing Reserved Memory. Backups are cross-compatible, and can be restored into an ElastiCache Serverless cache or a self-designed cluster.

Feature	Serverless caching	Self-designed clusters
Monitoring	Support cache level metrics including cache hit rate, cache miss rate, data size, and ECPUs consumed. 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].	ElastiCache self-designed clusters emit metrics at each node level, including both host-level metrics and cache metrics. Self-designed clusters emit SNS notifications for significa nt events. See [Metrics for Redis OSS].
Availability	99.99% availability [Service Level Agreement (SLA)]	Self-designed clusters can be designed to achieve up to 99.99% availability [Service Level Agreement (SLA)], depending on the configura tion.
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.	Self-designed clusters offer customer-enabled self-serv ice for minor and patching version upgrades, as well as major version upgrades. Managed updates are automatically applied during customer defined maintenan ce windows. Customers can also choose to apply a minor or patch version upgrade on-demand.

Feature	Serverless caching	Self-designed clusters
Global Data Store	Not supported	Supports Global Data Store, which enables cross region replication with single region writes and multi-region reads
Data Tiering	Not supported	Clusters that are designed using nodes from the r6gd family have their data tiered between memory and local SSD (solid state drives) storage. Data tiering provides a price-performance option for Redis OSS workloads by utilizing lower-cost solid state drives (SSDs) in each cluster node, in addition to storing data in memory.
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:

- Designing and managing your own ElastiCache (Redis OSS) cluster

Amazon ElastiCache resources

We recommend that you begin by reading the following sections, and refer to them as you need them:

- **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 (Redis OSS) section includes information on creating a cache cluster. It also includes how to authorize access to the cache cluster, connect to a cache node, and delete the cache cluster.
- **Performance at scale –** The Performance at scale with Amazon ElastiCache whitepaper addresses caching strategies that help your application to perform well at scale.

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 (Redis OSS) with Multi-AZ.

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

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 the following:

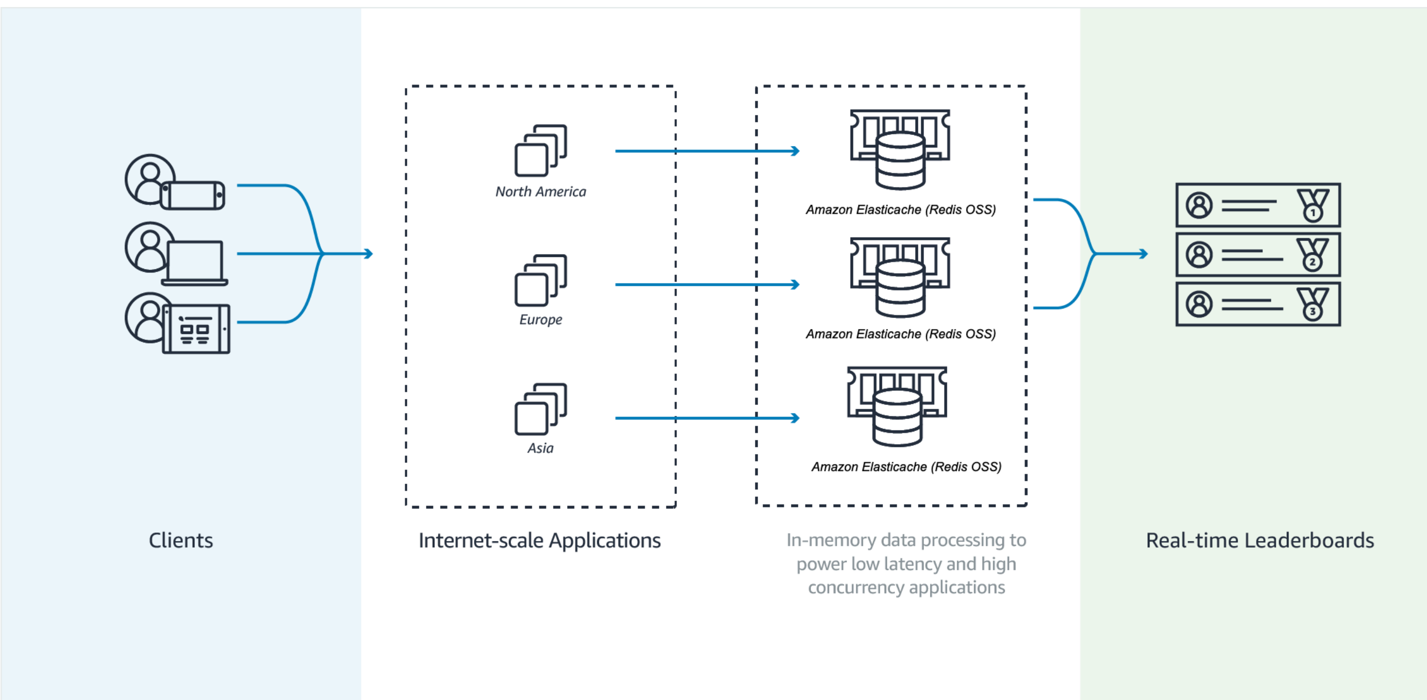
- Caching Strategies in the *ElastiCache (Redis OSS) User Guide*

Gaming Leaderboards (Redis OSS Sorted Sets)

Redis OSS sorted sets move the computational complexity of leaderboards from your application to your Redis OSS 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. Redis OSS sorted sets guarantee both uniqueness and element ordering. Using Redis OSS 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 (Redis OSS) gaming leaderboard works.



Example 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.

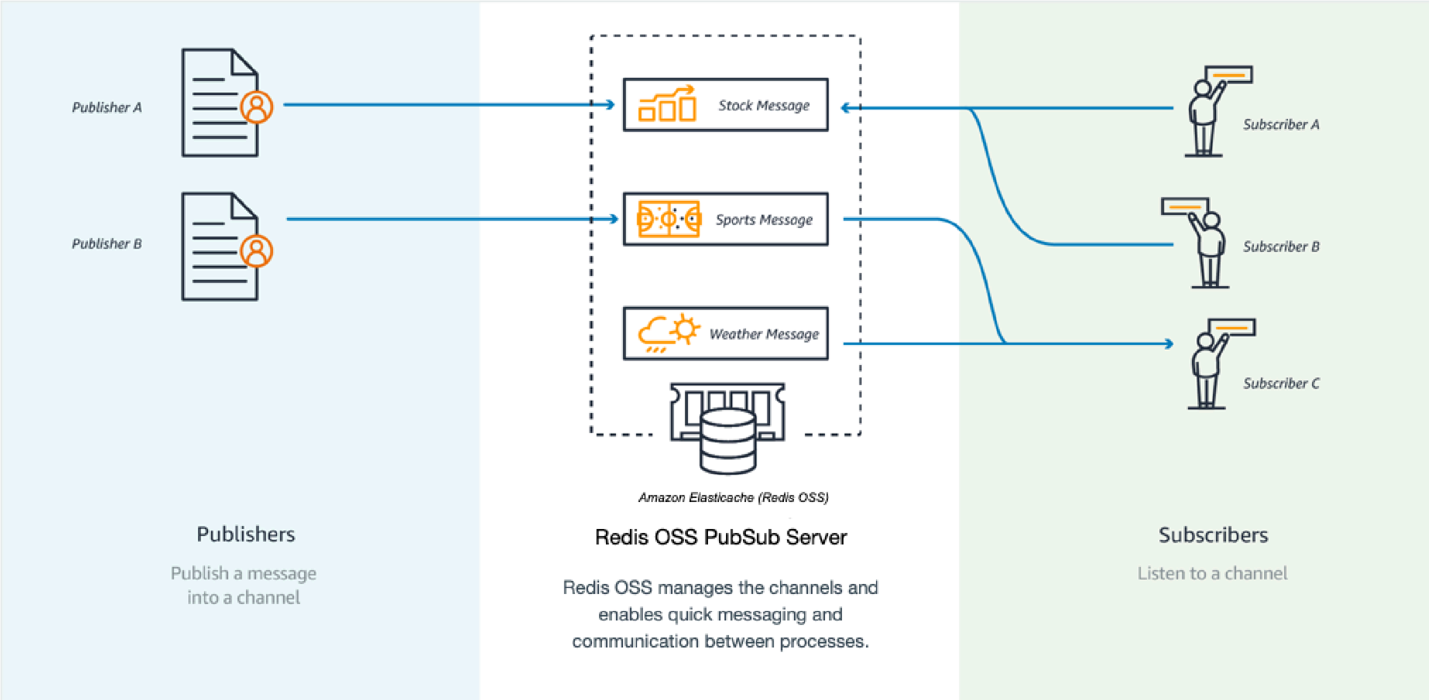
```
ZREVRANK leaderboard June
1
```

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

Messaging (Redis Pub/Sub)

When you send an email message, you send it to one or more specified recipients. In the 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*.

Redis 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 (Redis OSS) messaging.



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*.

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's subscribed to.

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

Recommendation Data (Redis Hashes)

Using INCR or DECR in Redis 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 Redis 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
```

Other Redis Uses

The blog post How to take advantage of Redis just adding it to your stack by Salvatore Sanfilippo discusses a number of common database concerns and how they can be easily solved using Redis OSS. This approach removes load from your database and improves performance.

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.

Getting started with Amazon ElastiCache (Redis OSS)

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

Topics

- Setting up
- Step 1: Create a cache
- Step 2: Read and write data to the cache
- Step 3: (Optional) Clean up
- Next Steps
- Getting Started with ElastiCache and AWS SDKs
- Tutorial: Configuring a Lambda function to access Amazon ElastiCache in an Amazon VPC

Setting up

To set up ElastiCache:

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Grant programmatic access
- 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 redis-cli

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

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

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

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

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

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use.

Which user needs programmatic access?	To	By
		• For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Related topics:

- What is IAM in the *IAM User Guide*.
- AWS Security Credentials in *AWS General Reference*.

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 Creating 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 Creating 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: Getting started with EC2.
- 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 self-designed clusters use port 6379 for Redis OSS commands, and ElastiCache serverless uses both port 6379 and port 6380. In order to successfully connect and execute Redis OSS commands from your EC2 instance, your security group must allow access to these ports as needed.

1. Sign in to the AWS Command Line Interface and open the [Amazon EC2 console](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:
    a. Choose **Edit**.
    b. Choose **Add rule**.
    c. In the Type column, choose **Custom TCP rule**.
    d. In the **Port range** box, type `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 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. In the **Port range**box, type `6380`.
    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 redis-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](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html).
2. Download and install redis-cli utility by running the appropriate command for your setup.

Amazon Linux 2023

```
sudo yum install redis6 -y
```

Amazon Linux 2

```
sudo amazon-linux-extras install epel -y
sudo yum install gcc jemalloc-devel openssl-devel tcl tcl-devel -y
sudo wget http://download.redis.io/redis-stable.tar.gz
sudo tar xvzf redis-stable.tar.gz
cd redis-stable
sudo make BUILD_TLS=yes
```

> **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 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.

Step 1: Create a 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 https://console.aws.amazon.com/connect/.
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 Step 2: Read and write data to the cache.

Step 2: 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 redis-cli on your EC2 instance. For more information on that step see Setting up.

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 **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 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 *redis-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/redis-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 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 *redis-cli* utility to connect to a cluster. The following command connects to a cache. Open the Command Prompt and change to the Redis OSS directory and run the command (note: replace Cache_Endpoint with the endpoint you retrieved in the previous step).

```
c:\Redis>redis-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 Step 3: (Optional) Clean up.

Step 3: (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 **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 (Redis OSS)
- Logging and monitoring in Amazon ElastiCache
- ElastiCache best practices and caching strategies
- Snapshot and restore
- Amazon SNS monitoring of ElastiCache events

Getting Started with ElastiCache and AWS SDKs

This section contains hands-on tutorials to help you learn about Amazon ElastiCache. 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.

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 clusters (cluster mode enabled and cluster mode disabled)
- Check if users or user groups exist, otherwise create them (Redis OSS 6.0 onwards only)
- 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

Topics

Tutorial Prerequisites

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

Creating ElastiCache clusters and users

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

Topics

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
 cache 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/red-ug/
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='redis',
        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, Redis OSS 6, one primary and two replicas
    elasticacheResponse = create_cluster_mode_disabled(
        #CacheNodeType='cache.m6g.large',
        EngineVersion='6.0',
        NumCacheClusters=3,
        ReplicationGroupDescription='Redis cluster mode disabled with replicas',
        ReplicationGroupId='redis202104053'
        )

    logging.info(elasticacheResponse)
```

To run the program, enter the following command:

```
python CreateClusterModeDisabledCluster.py
```

For more information, see Managing clusters.

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 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
 cache 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/red-ug/
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': 'redis',
            '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, Redis OSS 6, 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='6.0',
        NumCacheClusters=3,
        ReplicationGroupDescription='Redis OSS cluster mode disabled with replicas',
        ReplicationGroupId='redis202104',
        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.

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/red-ug/
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='redis',
        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.redis6.0.cluster.on'
    )

    return response


# Creates a cluster mode enabled
response = create_cluster_mode_enabled(
    CacheNodeType='cache.m6g.large',
    EngineVersion='6.0',
    ReplicationGroupDescription='Redis OSS cluster mode enabled with replicas',
    ReplicationGroupId='redis20210',
#   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.

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 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.redis6.0.clus
    """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/red-ug/
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': 'redis',
            '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='6.0',
        ReplicationGroupDescription='Redis OSS cluster mode enabled with replicas',
        ReplicationGroupId='redis2021',
    #   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.

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 (Redis OSS) replication groups. By doing this, you can establish security boundaries between clients using the same 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.

```
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. For
 details refer to https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/
Clusters.RBAC.html#Access-string
    :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`

Connecting to ElastiCache

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

Topics

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

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

```
from rediscluster import RedisCluster
import logging
```

```
logging.basicConfig(level=logging.INFO)
redis = RedisCluster(startup_nodes=[{"host":
 "xxx.yyy.clustercfg.zzz1.cache.amazonaws.com","port": "6379"}],
 decode_responses=True,skip_full_coverage_check=True)

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

To run the program, enter the following command:

```
python ConnectClusterModeEnabled.py
```

Usage examples

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

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 a Lambda function to access Amazon ElastiCache in an Amazon 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: Create a serverless cache

To create a serverless cache, follow these steps.

Topics

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 Step 1: Create a cache.

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

Note that the value of the Status field is set to `CREATING`. It can take a minute for ElastiCache to finish creating your cache.

Step 1.2: Copy serverless cache endpoint

Verify that ElastiCache (Redis OSS) 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 your account to assume the new role. Save the policy to a file named *trust-policy.json*.

```
{
"Version": "2012-10-17",
    "Statement": [{
"Effect": "Allow",
        "Principal": { "AWS": "arn:aws:iam::123456789012:root" },
        "Action": "sts:AssumeRole"
    },
    {
      "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: Create a serverless cache

1. Create a new default user.

```
aws elasticache create-user \
  --user-name default \
--user-id default-user-disabled \
--engine redis \
--authentication-mode Type=no-password-required \
--access-string "off +get ~keys*"
```

2. 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"
```

3. 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-user-disabled 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

To create a Lambda function, take these steps.

Topics

- Step 2.1: Create a Lambda function
- Step 2.2: Create the IAM role (execution role)
- Step 2.3: Upload the deployment package (create the Lambda function)

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 step 1.2.

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

import botocore.session
```

```
import redis
from botocore.model import ServiceId
from botocore.signers import RequestSigner
from cachetools import TTLCache, cached
import uuid

class ElastiCacheIAMProvider(redis.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)
    redis_client = redis.Redis(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
    redis_client.set(key, uuid_in)
    result = redis_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 Redis OSS.")
    else:
        raise Exception(f"Bad value retrieved. Expected {uuid_in}, got
 {decoded_result}")

    return "Fetched value from Redis"
```

This code uses the Python redis-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 redis-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 redis-py and cachetools libraries into.

```
mkdir package
```

Install redis-py, cachetools using pip.

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

Create a .zip file containing the redis-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 redis-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 (AccessRedis) 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 step 2.2. 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 AccessRedis  \
--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

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 (AccessRedis) using the AWS Lambda invoke command.

   ```
   aws lambda invoke \
   --function-name AccessRedis  \
   --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/AccessRedis). The log stream should contain output similar to the following:

     ```
     Success: Inserted 826e70c5f4d2478c8c18027125a3e01e. Fetched
      826e70c5f4d2478c8c18027125a3e01e from Redis OSS.
     ```

   - Review the results in the AWS Lambda console.

Step 4: Clean up (Optional)

To clean up, take these steps.

Topics

Step 4.1: Delete Lambda function

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

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"
```

Designing and managing your own ElastiCache (Redis OSS) cluster

If you need fine-grained control over your ElastiCache cluster, you can choose to design your own 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. For details on managing, updating or deleting nodes or clusters, see Managing nodes. For an overview of the major components of an Amazon ElastiCache deployment when you design your own ElastiCache cluster, see these key concepts.

Topics

ElastiCache components and features

Following, you can find an overview of the major components of an Amazon ElastiCache deployment.

Topics

- AWS Regions and availability zones
- ElastiCache endpoints
- ElastiCache parameter groups
- ElastiCache security
- ElastiCache subnet groups
- ElastiCache (Redis OSS) 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 (Redis OSS) .

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 ElastiCache reserved nodes.

For more information on nodes, see Managing nodes.

ElastiCache (Redis OSS) shards

A Redis OSS *shard* (called a *node group* in the API and CLI) is a grouping of one to six related nodes. A 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.

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 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.

ElastiCache (Redis OSS) clusters

A Redis OSS *cluster* is a logical grouping of one or more ElastiCache (Redis OSS) shards. Data is partitioned across the shards in a 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 and Managing nodes

  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 replication groups.

Typical cluster configurations

Following are typical cluster configurations.

Redis OSS clusters

Redis OSS clusters with cluster mode disabled always contain just one shard (in the API and CLI, one node group). A 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 Redis OSS cluster and enabling Multi-AZ. For more information, see Mitigating Failures.

As demand upon your 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 Redis OSS 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.

ElastiCache supports changing a Redis OSS cluster's node type to a larger node type dynamically. For information on scaling up or down, see Scaling single-node clusters for Redis OSS (Cluster Mode Disabled) or Scaling Redis OSS (Cluster Mode Disabled) clusters with replica nodes.

ElastiCache replication

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.

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.

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: Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled)--> Redis OSS cluster with cluster mode enabled vs. Redis OSS cluster with cluster mode disabled

In the following table, you can find a comparison of the features of Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled) replication groups.

	Redis OSS cluster with cluster mode disabled	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

	Redis OSS cluster with cluster mode disabled	Redis OSS cluster with cluster mode enabled
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 Redis OSS (cluster mode enabled).		
You can use backup and restore to resize your 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.

Replication: Limits and exclusions

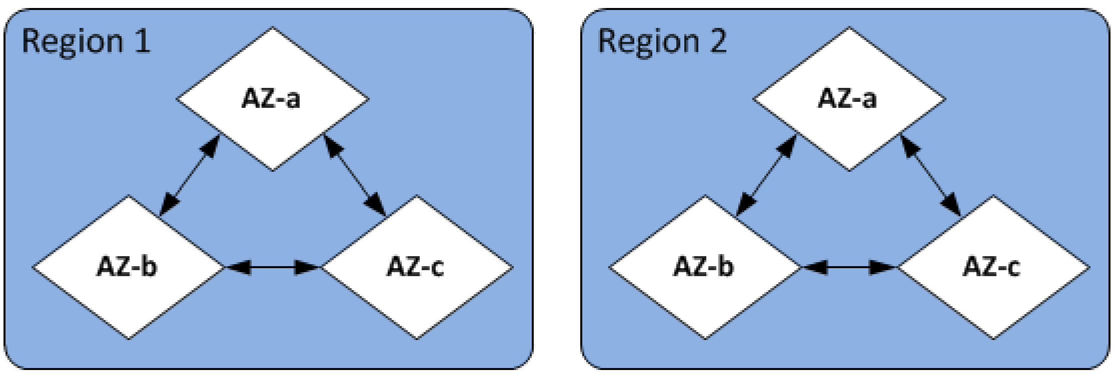
- Multi-AZ is not supported on node types T1.

AWS Regions and availability zones

Amazon ElastiCache is available in multiple AWS Regions around the world. Thus, you can launch ElastiCache clusters in the locations that meet your business requirements. For example, you can launch in the AWS Region closest to your customers or to meet certain legal requirements.

By default, the AWS SDKs, AWS CLI, ElastiCache API, and ElastiCache console reference the US West (Oregon) Region. As ElastiCache expands availability to new AWS Regions, new endpoints for these AWS Regions are also available. You can use these in your HTTP requests, the AWS SDKs, AWS CLI, and ElastiCache console.

Each AWS Region is designed to be completely isolated from the other AWS Regions. Within each are multiple Availability Zones. By launching your nodes in different Availability Zones, you can achieve the greatest possible fault tolerance. For more information about AWS Regions and Availability Zones, see Choosing regions and availability zones. In the following diagram, you can see a high-level view of how AWS Regions and Availability Zones work.



For information on AWS Regions supported by ElastiCache and their endpoints, see Supported regions & endpoints.

ElastiCache endpoints

An *endpoint* is the unique address your application uses to connect to an ElastiCache node or cluster.

Single node endpoints for Redis OSS with cluster mode disabled

The endpoint for a single node Redis OSS cluster is used to connect to the cluster for both reads and writes.

Multi-node endpoints for Redis OSS with cluster mode disabled

A multiple node 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).

Redis OSS (Cluster Mode Enabled) endpoints

A 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.

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 parameter groups.

ElastiCache security

For enhanced security, ElastiCache (Redis OSS) 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 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 (Redis OSS) backups

A *backup* is a point-in-time copy of a Redis OSS cluster. 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 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
- How synchronization and backup are implemented
- Performance impact of backups of self-designed clusters
- Ensuring that you have enough memory to create a Redis OSS snapshot

ElastiCache events

When important events happen on a cache 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 (Redis OSS) terminology

In October 2016, Amazon ElastiCache launched support for Redis OSS 3.2. At that point, we added support for partitioning your data across up to 500 shards (called node groups in the ElastiCache API and AWS CLI). To preserve compatibility with previous versions, we extended API version 2015-02-02 operations to include the new Redis OSS functionality.

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 cache 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.

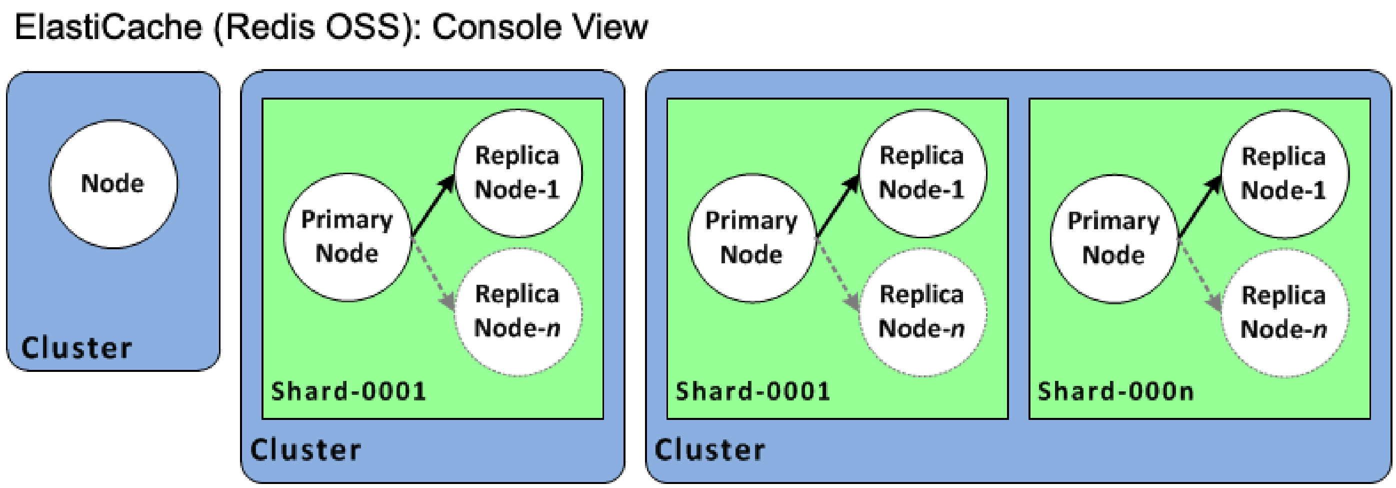
Cluster vs. replication group

The console now uses the term *cluster* for all ElastiCache (Redis OSS) clusters. The console uses the term cluster in all these circumstances:

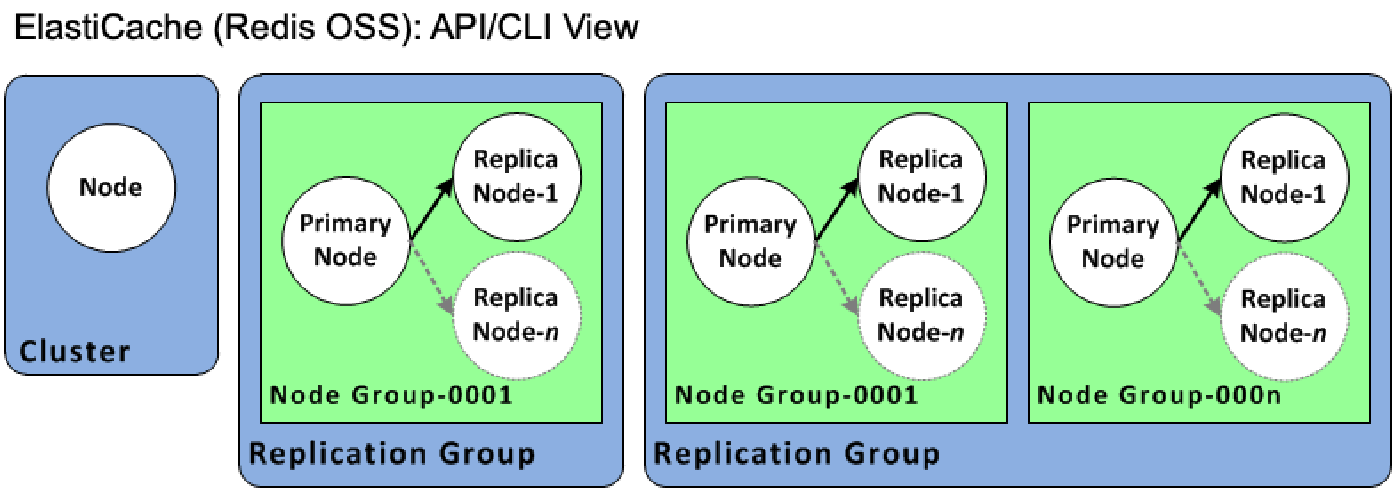
- When the cluster is a single node Redis OSS cluster.
- When the cluster is a Redis OSS (cluster mode disabled) cluster that supports replication within a single shard (in the API and CLI, called a *node group*).
- When the cluster is a Redis OSS (cluster mode enabled) cluster that supports replication 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 replication groups, see High availability using replication groups.

The following diagram illustrates the various topologies of ElastiCache (Redis OSS) clusters from the console's perspective.



The ElastiCache API and AWS CLI operations still distinguish single node ElastiCache (Redis OSS) clusters from multi-node replication groups. The following diagram illustrates the various ElastiCache (Redis OSS) topologies from the ElastiCache API and AWS CLI perspective.



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 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].

Designing your own cluster

Following are the one-time actions you must take to start designing your ElastiCache cluster.

Topics

Setting up 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 cache clusters in a VPC. When launching a cache 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.

Step 1: Create a subnet group

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)

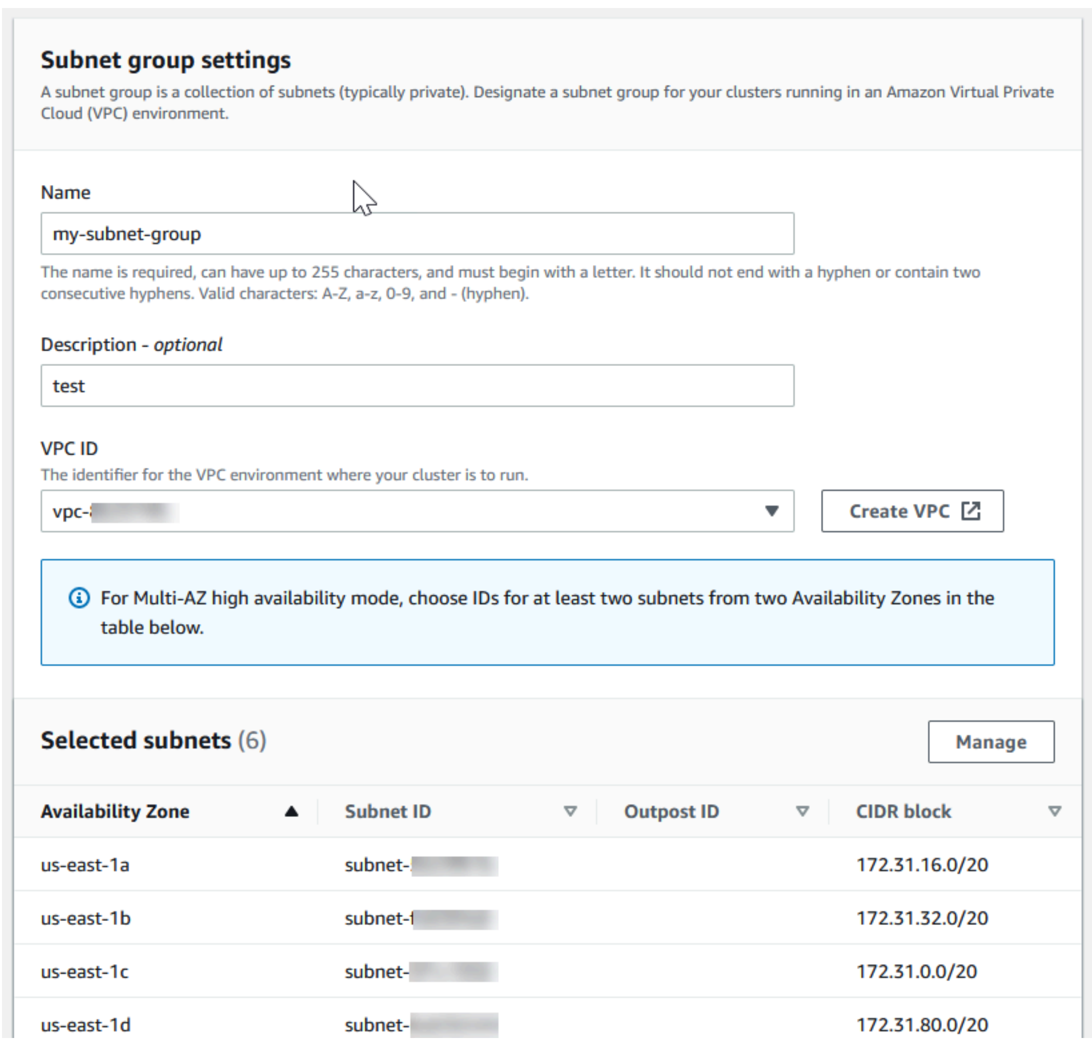
1. Sign in to the AWS Management Console, and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. In the navigation list, choose **Subnet Groups**.
3. Choose **Create Subnet Group**.
4. In the **Create Subnet Group** wizard, do the following. When all the settings are as you 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 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 **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 (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 [Redis OSS-specific parameters](#) and [Creating a 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 an r6gd node type, data-tiering is automatically enabled. For more information, see [Data tiering](#).

      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.
      - 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.

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 Redis OSS engine version 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.
  - **Redis OSS AUTH Default User** – An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.
- **Redis OSS AUTH** – An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.

> **Note**
>
> For Redis OSS versions between 3.2.6 onward, excluding version 3.2.10, Redis OSS 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 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 (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.

Creating a Redis OSS (cluster mode disabled) cluster (AWS CLI)

Example

The following CLI code creates a Redis OSS (cluster mode disabled) cache 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 Redis OSS (cluster mode enabled) cluster (Console).
- To use the AWS CLI, see Creating a 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 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.

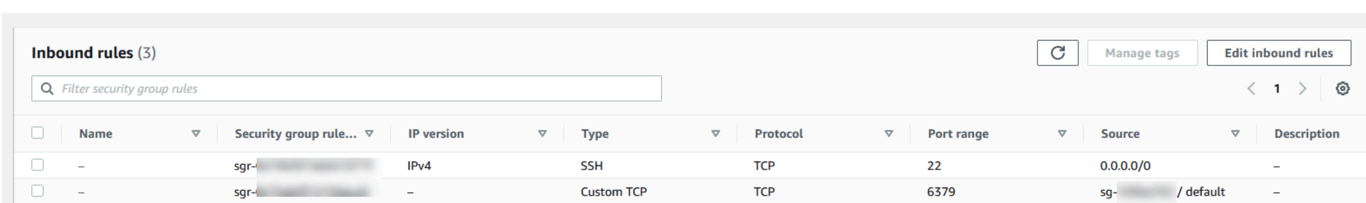
To create a rule in a VPC security group that allows connections from another security group

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 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 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 (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 (Redis OSS) cluster’s multiple read replicas in different AWS Availability Zones (AZ) to ensure high availability of reader endpoints.

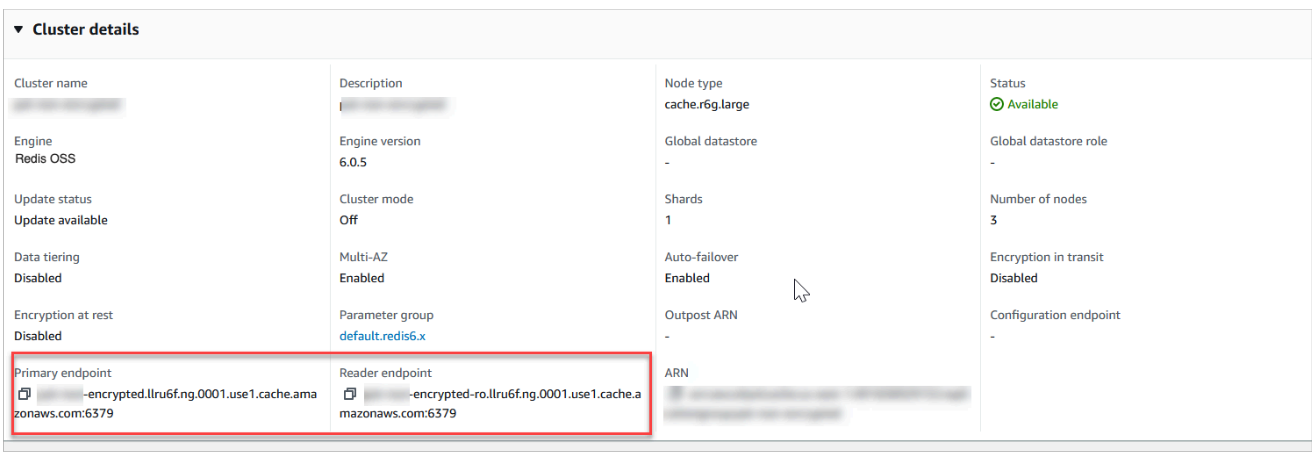
> **Note**
>
> A reader endpoint is not a load balancer. It is a DNS record that will resolve to an IP address of one of the replica nodes in a round robin fashion.

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**.

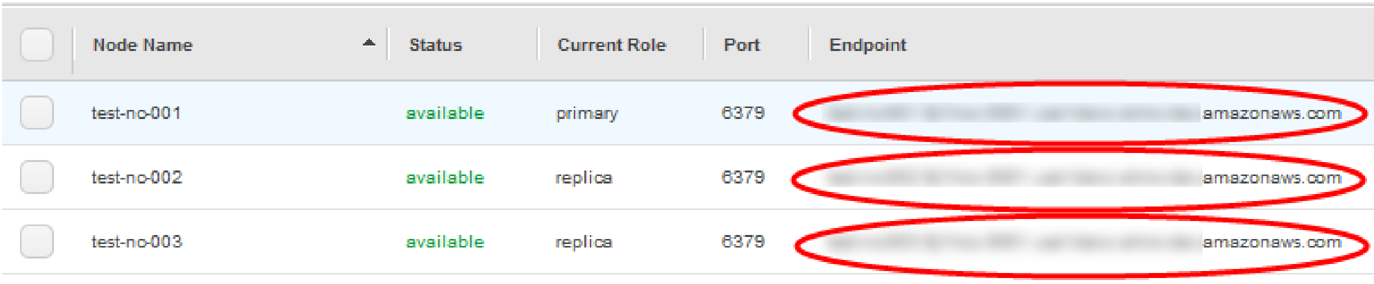
   The clusters screen will appear with a list that will include any existing Redis OSS serverless caches, Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled) clusters. Choose the cluster you created in the [Creating a Redis OSS (cluster mode disabled) cluster (Console)]() section.
3. 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 Name	Status	Current Role	Port	Endpoint
test-nc-001	available	primary	6379	amazonaws.com
test-nc-002	available	replica	6379	amazonaws.com
test-nc-003	available	replica	6379	amazonaws.com

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
- Finding Endpoints for a 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 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 *redis-cli* utility to connect to a cluster. The latest version of redis-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 redis-cli with other Linux distributions, see the documentation for your specific operating system..

Note

This process covers testing a connection using redis-cli utility for unplanned use only. For a list of supported Redis OSS clients, see the Redis OSS documentation. For examples of using the AWS SDKs with ElastiCache, see Getting Started with ElastiCache and AWS SDKs.

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 Redis OSS is 6379.)

   ```
   src/redis-cli -h primary-endpoint -p port number
   ```

   The result in a Redis OSS command prompt looks similar to the following:

   ```
   primary-endpoint:port number
   ```

2. You can now run 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 Redis OSS is 6379.)

   ```
   src/redis-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 Redis OSS command prompt looks similar to the following:

   ```
   configuration-endpoint:port number
   ```

2. You can now run 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, redis-cli uses an unencrypted TCP connection when connecting to Redis OSS. The option `BUILD_TLS=yes` enables SSL/TLS at the time of redis-cli compilation as shown in the preceding Download and set up redis-cli 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 redis-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/redis-cli -h cluster-endpoint --tls -a your-password -p port number
```

The following example connects to a cluster that has only encryption enabled:

```
src/redis-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/redis-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/redis-cli -c -h cluster-endpoint --tls -p port number
```

After you connect to the cluster, you can run the Redis OSS commands as shown in the preceding examples for unencrypted clusters.

Redis-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 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 Redis OSS cluster or replication group (Windows)

In order to connect to the Redis OSS cluster from an EC2 Windows instance using the Redis OSS CLI, you must download the *redis-cli* package and use *redis-cli.exe* to connect to the Redis OSS cluster from an EC2 Windows instance.

In the following example, you use the *redis-cli* utility to connect to a cluster that is not encryption enabled and running Redis OSS. For more information about Redis OSS and available Redis OSS commands, see Redis OSS commands on the Redis OSS website.

To connect to a Redis OSS cluster that is not encryption-enabled using *redis-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 Redis OSS directory and run the command `c:\Redis>redis-cli -h Redis_Cluster_Endpoint -p 6379`.

   For example:

```
c:\Redis>redis-cli -h cmd.xxxxxxx.ng.0001.usw2.cache.amazonaws.com -p 6379
```

3. Run Redis OSS commands.

   You are now connected to the cluster and can run 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 redis-cli
```

Step 5: 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 (Redis OSS) 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.

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 console dashboard, choose Redis OSS.

   A list of all caches running Redis OSS 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 cache 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 cache cluster. To delete multiple cache clusters, call `delete-cache-cluster` for each cache cluster that you want to delete. You do not need to wait for one cache 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`.

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

Following, you can find videos to help you learn basic and advanced Amazon ElastiCache concepts. For information about AWS Training, see AWS Training & Certification.

Topics

Introductory Videos

The following videos introduce you to Amazon ElastiCache.

Topics

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 (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 (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 (Redis OSS) online resharding
- ElastiCache security and encryption
- ElastiCache (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 (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 (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 (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 Memcached and 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 Memcached and 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).

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.

Managing nodes

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 or replication group 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.

Generally speaking, due to its support for sharding, Redis OSS (cluster mode enabled) deployments have a number of smaller nodes. In contrast, 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

Some important operations involving nodes are the following:

- Adding nodes to a cluster
- Removing nodes from a cluster
- Scaling ElastiCache (Redis OSS)
- Finding connection endpoints

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 inaccessi ble 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: • 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. • The VPC security groups are updating for the ElastiCac he Cluster. • The ElastiCache cluster is being scaled up or scaled down.

Type	Billed	Description
		• The log delivery configurations are being modified for the ElastiCache Cluster. • A delete operation for the ElastiCache node is pending. • The ElastiCache (Redis OSS) password is being updated/rotated using AWS Secrets Manager.
`rebooting cache 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.
`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.

Type	Billed	Description
`incompatible_network`	Not billed	An incompatible-network state indicates one or more of the following is true of the Amazon ElastiCache node: • There are no available IP addresses in the subnet that the ElastiCache node was launched into. • 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: • 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. • The specified RDB snapshot failed to restore. • The AWS account for the ElastiCache cluster has been suspended. • The node failed and could not be recovered.
snapshotting	Billed	ElastiCache is creating a snapshot of the ElastiCache (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
```

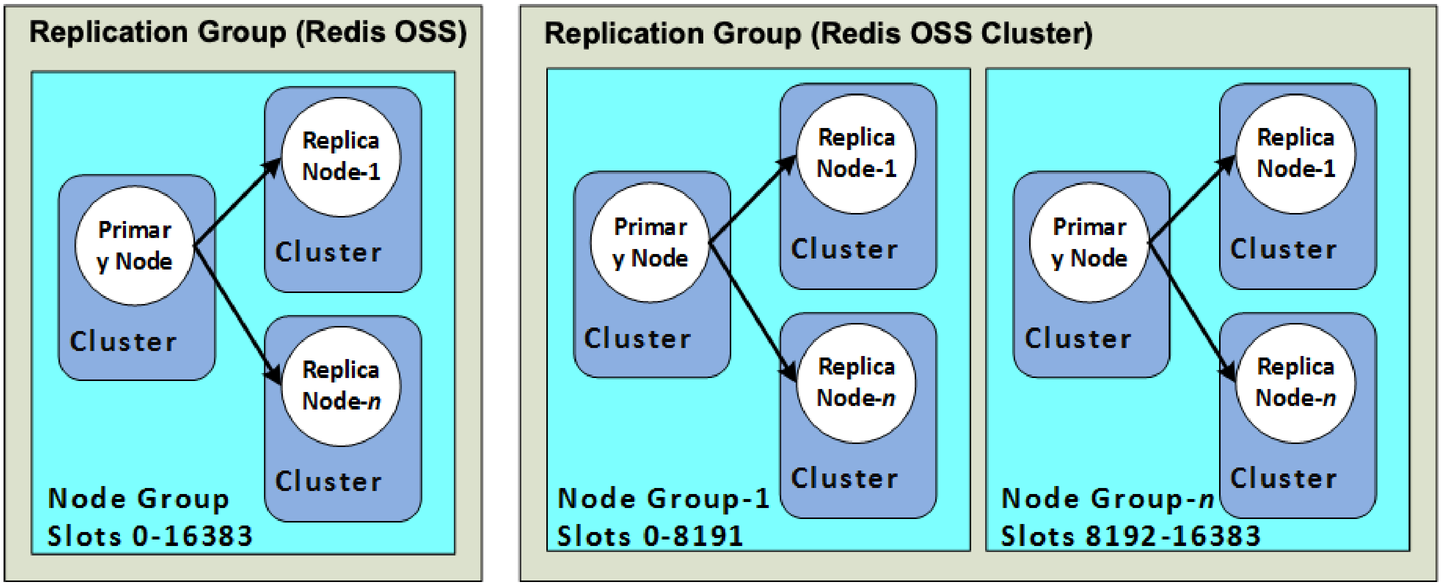
Viewing ElastiCache Node Status through the API

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.

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. Redis OSS version 3.2 and later support multiple shards within a cluster (in the API and CLI, a replication group). This support enables partitioning your data in a Redis OSS (cluster mode enabled) cluster.

The following diagram illustrates the differences between a Redis OSS (cluster mode disabled) cluster and a Redis OSS (cluster mode enabled) cluster.



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 Redis OSS cluster, you specify whether you want to create a cluster with clustering enabled. 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 Redis OSS (Cluster Mode Disabled) replication groups or Deleting a read replica, for Redis OSS (Cluster Mode Disabled) replication groups . Redis OSS (cluster mode disabled) clusters can also scale vertically by changing node types. For more information, see Scaling Redis OSS (Cluster Mode Disabled) clusters with replica nodes.

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**.

After a Redis OSS (cluster mode enabled) cluster is created, it can be altered (scaled in or out). For more information, see Scaling ElastiCache (Redis OSS) and Replacing nodes.

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

Before attempting to connect to the nodes in your Redis OSS cluster, you must have the endpoints for the nodes. To find the endpoints, see the following:

- Finding a Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding Endpoints for a Redis OSS (Cluster Mode Enabled) Cluster (Console)
- Finding Endpoints (AWS CLI)
- Finding Endpoints (ElastiCache API)

In the following example, you use the *redis-cli* utility to connect to a cluster that is running Redis OSS.

> **Note**
>
> For more information about Redis OSS and available Redis OSS commands, see the http://redis.io/commands webpage.

To connect to a Redis OSS cluster using the *redis-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 `redis-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 *redis-cli* utility. This utility is included in the Redis OSS 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 http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make distclean      # ubuntu systems only
make
```

4. At the command prompt of your EC2 instance, type the following command.

```
src/redis-cli -c -h mycachecluster.eaogs8.0001.usw2.cache.amazonaws.com -p 6379
```

A Redis OSS command prompt similar to the following appears.

```
redis mycachecluster.eaogs8.0001.usw2.cache.amazonaws.com 6379>
```

5. Test the connection by running Redis OSS commands.

   You are now connected to the cluster and can run Redis OSS commands. The following are some example commands with their 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 redis-cli
```

For connecting to nodes or clusters which have Secure Sockets Layer (SSL) encryption (in-transit enabled), see ElastiCache in-transit encryption (TLS).

Supported node types

ElastiCache supports the following node types. Generally speaking, the current generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For more information on performance details for each node type, see Amazon EC2 Instance Types.

For information on which node size to use, see Choosing your node size.

Current Generation

For more information on Previous Generation, please refer to Previous Generation Nodes.

> **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.

General

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.m7g.large	6.2	N	N	N	0.937	12.5
cache.m7g.xlarge	6.2	Y	Y	Y	1.876	12.5
cache.m7g.2xlarge	6.2	Y	Y	Y	3.75	15
cache.m7g.4xlarge	6.2	Y	Y	Y	7.5	15

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.m7g.8xlarge	6.2	Y	Y	Y	15	N/A
cache.m7g.12xlarge	6.2	Y	Y	Y	22.5	N/A
cache.m7g.16xlarge	6.2	Y	Y	Y	30	N/A
cache.m6g.large	5.0.6	N	N	N	0.75	10.0
cache.m6g.xlarge	5.0.6	Y	Y	Y	1.25	10.0
cache.m6g.2xlarge	5.0.6	Y	Y	Y	2.5	10.0
cache.m6g.4xlarge	5.0.6	Y	Y	Y	5.0	10.0
cache.m6g.8xlarge	5.0.6	Y	Y	Y	12	N/A
cache.m6g.12xlarge	5.0.6	Y	Y	Y	20	N/A
cache.m6g.16xlarge	5.0.6	Y	Y	Y	25	N/A
cache.m5.large	3.2.4	N	N	N	0.75	10.0

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.m5.xlarge	3.2.4	Y	N	N	1.25	10.0
cache.m5.2xlarge	3.2.4	Y	Y	Y	2.5	10.0
cache.m5.4xlarge	3.2.4	Y	Y	Y	5.0	10.0
cache.m5.12xlarge	3.2.4	Y	Y	Y	12	N/A
cache.m5.24xlarge	3.2.4	Y	Y	Y	25	N/A
cache.m4.large	3.2.4	N	N	N	0.45	1.2
cache.m4.xlarge	3.2.4	Y	N	N	0.75	2.8
cache.m4.2xlarge	3.2.4	Y	Y	Y	1.0	10.0
cache.m4.4xlarge	3.2.4	Y	Y	Y	2.0	10.0
cache.m4.10xlarge	3.2.4	Y	Y	Y	5.0	10.0
cache.t4g.micro	3.2.4	N	N	N	0.064	5.0
cache.t4g.small	5.0.6	N	N	N	0.128	5.0

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.t4g.medium	5.0.6	N	N	N	0.256	5.0
cache.t3.micro	3.2.4	N	N	N	0.064	5.0
cache.t3.small	3.2.4	N	N	N	0.128	5.0
cache.t3.medium	3.2.4	N	N	N	0.256	5.0
cache.t2.micro	3.2.4	N	N	N	0.064	1.024
cache.t2.small	3.2.4	N	N	N	0.128	1.024
cache.t2.medium	3.2.4	N	N	N	0.256	1.024

Memory optimized

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.r7g.large	6.2	N	N	N	0.937	12.5
cache.r7g.xlarge	6.2	Y	Y	Y	1.876	12.5

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.r7g.2xlarge	6.2	Y	Y	Y	3.75	15
cache.r7g.4xlarge	6.2	Y	Y	Y	7.5	15
cache.r7g.8xlarge	6.2	Y	Y	Y	15	N/A
cache.r7g.12xlarge	6.2	Y	Y	Y	22.5	N/A
cache.r7g.16xlarge	6.2	Y	Y	Y	30	N/A
cache.r6g.large	5.0.6	N	N	N	0.75	10.0
cache.r6g.xlarge	5.0.6	Y	Y	Y	1.25	10.0
cache.r6g.2xlarge	5.0.6	Y	Y	Y	2.5	10.0
cache.r6g.4xlarge	5.0.6	Y	Y	Y	5.0	10.0
cache.r6g.8xlarge	5.0.6	Y	Y	Y	12	N/A
cache.r6g.12xlarge	5.0.6	Y	Y	Y	20	N/A

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.r6g.16xlarge	5.0.6	Y	Y	Y	25	N/A
cache.r5.large	3.2.4	N	N	N	0.75	10.0
cache.r5.xlarge	3.2.4	Y	N	N	1.25	10.0
cache.r5.2xlarge	3.2.4	Y	Y	Y	2.5	10.0
cache.r5.4xlarge	3.2.4	Y	Y	Y	5.0	10.0
cache.r5.12xlarge	3.2.4	Y	Y	Y	12	N/A
cache.r5.24xlarge	3.2.4	Y	Y	Y	25	N/A
cache.r4.large	3.2.4	N	N	N	0.75	10.0
cache.r4.xlarge	3.2.4	Y	N	N	1.25	10.0
cache.r4.2xlarge	3.2.4	Y	Y	Y	2.5	10.0
cache.r4.4xlarge	3.2.4	Y	Y	Y	5.0	10.0
cache.r4.8xlarge	3.2.4	Y	Y	Y	12	N/A
cache.r4.16xlarge	3.2.4	Y	Y	Y	25	N/A

Memory optimized with data tiering

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.r6gd.xlarge	6.2.0	Y	N	N	1.25	10
cache.r6gd.2xlarge	6.2.0	Y	Y	Y	2.5	10
cache.r6gd.4xlarge	6.2.0	Y	Y	Y	5.0	10
cache.r6gd.8xlarge	6.2.0	Y	Y	Y	12	N/A
cache.r6gd.12xlarge	6.2.0	Y	Y	Y	20	N/A
cache.r6gd.16xlarge	6.2.0	Y	Y	Y	25	N/A

Network optimized

Instance type	Minimum supported Redis OSS version	Enhanced I/O (Redis OSS 5.0.6+)	TLS Offloading (Redis OSS 6.2.5+)	Enhanced I/O Multiplexing (Redis OSS 7.0.4+)	Baseline bandwidth (Gbps)	Burst bandwidth (Gbps)
cache.c7gn.large	6.2	N	N	N	6.25	30
cache.c7gn.xlarge	6.2	Y	Y	Y	12.5	40
cache.c7gn.2xlarge	6.2	Y	Y	Y	25	50
cache.c7gn.4xlarge	6.2	Y	Y	Y	50	N/A
cache.c7gn.8xlarge	6.2	Y	Y	Y	100	N/A
cache.c7gn.12xlarge	6.2	Y	Y	Y	150	N/A
cache.c7gn.16xlarge	6.2	Y	Y	Y	200	N/A

Supported node types by AWS Region

Supported node types may vary between AWS Regions. For more details, see Amazon ElastiCache pricing.

Burstable Performance Instances

You can launch general-purpose burstable T4g, T3-Standard and T2-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, T3 and T2 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	Memory (GiB)	Network performance
12	288	2	10%	0.5	Up to 5 Gigabit
24	576	2	20%	1.37	Up to 5 Gigabit
24	576	2	20%	3.09	Up to 5 Gigabit
12	288	2	10%	0.5	Up to 5 Gigabit
24	576	2	20%	1.37	Up to 5 Gigabit

CPU credits earned per hour	Maximum earned credits that can be accrued*	vCPUs	Baseline performance per vCPU	Memory (GiB)	Network performance
24	576	2	20%	3.09	Up to 5 Gigabit
6	144	1	10%	0.5	Low to moderate
12	288	1	20%	1.55	Low to moderate
24	576	2	20%	3.22	Low to moderate

* 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:

> **Note**
>
> These metrics are not available for T2 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 append-only files (AOF) aren't supported for T2 instances. Redis OSS configuration variables `appendonly` and `appendfsync` aren't supported on Redis OSS version 2.8.22 and later.

Related Information

- Amazon ElastiCache Product Features and Details
- Redis OSS-specific parameters
- In Transit Encryption (TLS)

Rebooting nodes (cluster mode disabled only)

Some changes require that cluster nodes be rebooted 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.

For Redis OSS (cluster mode disabled) clusters, those parameters are:

- activerehashing
- databases

You are able to reboot a node using only 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.

Redis OSS (Cluster Mode Enabled) parameter changes

If you make changes to the following parameters on a 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 Redis OSS (cluster mode enabled) cluster. See Deleting a cluster.
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 **Redis OSS**.

   A list of clusters running 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.

Replacing nodes

Amazon ElastiCache (Redis OSS) frequently upgrades its fleet with patches and upgrades being applied to instances seamlessly. However, from time to time we need to relaunch your ElastiCache (Redis OSS) 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 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.
- Redis OSS (cluster mode enabled) Configurations
  - Replace the only node in any Redis OSS cluster – A procedure to replace a node in a Redis OSS cluster using backup and restore.
  - Replace a replica node in any Redis OSS cluster – A procedure to replace a read-replica in any Redis OSS cluster by increasing and decreasing the replica count with no cluster downtime.
  - Replace any node in a Redis OSS (cluster mode enabled) shard – A dynamic procedure with no cluster downtime to replace a node in a Redis OSS (cluster mode enabled) cluster by scaling out and scaling in.
- Redis OSS (cluster mode disabled) Configurations
  - Replace the only node in any Redis OSS cluster – Procedure to replace any node in a Redis OSS cluster using backup and restore.

- Replace a replica node in any Redis OSS cluster – A procedure to replace a read-replica in any Redis OSS cluster by increasing and decreasing the replica count with no cluster downtime.
- Replace a node in a Redis OSS (cluster mode disabled) cluster – Procedure to replace a node in a Redis OSS (cluster mode disabled) cluster using replication.
- Replace a Redis OSS (cluster mode disabled) read-replica – A procedure to manually replace a read-replica in a Redis OSS (cluster mode disabled) replication group.
- Replace a Redis OSS (cluster mode disabled) primary node – A procedure to manually replace the primary node in a Redis OSS (cluster mode disabled) replication group.

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 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 (Redis OSS) provides improved availability during patching, updates, and other maintenance-related node replacements.

  For ElastiCache (Redis OSS) Cluster configurations that are set up to use ElastiCache (Redis OSS) Cluster clients, replacement now completes while the cluster serves incoming write requests.

  For non-Cluster configurations with autofailover enabled, clusters on 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 maintenance.

- **Replace the only node in any 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.
  4. In your application, replace the old node's endpoint with the new node's endpoint.

- **Replace a replica node in any 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 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 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 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 Redis OSS (cluster mode disabled) cluster** – If the cluster is a 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 Redis OSS cluster with no shards.
  2. Add a read-replica to the cluster. For instructions, see To add nodes to a cluster (console).
  3. Promote the newly created read-replica to primary. For instructions, see Promoting a read replica to primary, for Redis OSS (cluster mode disabled) replication groups.
  4. Delete the node scheduled for replacement. For instructions, see Removing nodes from a cluster.
  5. In your application, replace the old node's endpoint with the new node's endpoint.

- **Replace a 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 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 a 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 Redis OSS (Cluster Mode Disabled) replication groups
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 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 Redis OSS (cluster mode disabled) primary node

  1. Promote a read-replica to primary. For instructions, see Promoting a read replica to primary, for 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 a 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 Redis OSS (Cluster Mode Disabled) replication groups.
  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 .

ElastiCache reserved nodes

Reserving one or more nodes might be a way for you to reduce costs. Reserved nodes are charged an up front fee that depends upon the node type and the length of reservation— one or three years.

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 of 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.

> **Note**
>
> Reserved nodes are not flexible; they only apply to the exact instance type that you reserve.

Managing costs with reserved nodes

Reserving one or more nodes may be a way for you to reduce costs. 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.

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.

Topics

- Standard reserved node offerings
- Legacy reserved node offerings
- Getting info about reserved node offerings

- Purchasing a reserved node
- Getting info about your reserved nodes

Standard reserved node offerings

When you purchase a standard reserved node instance (RI) in Amazon ElastiCache, you 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.

The new node instance that you create must exactly match the specifications of the reserved node instance. If the specifications of the new 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 three 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.

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.

Offering	Up-front cost	Usage fee	Advantage
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.
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 Redis.
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, 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"
                }
            ]
}
```

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)

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 Redis.
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)

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 Redis OSS cluster

The following procedure describes how to migrate your Redis OSS cluster node type using the ElastiCache Console. During this process, your 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 Redis OSS 5.0.6 and above. All lower version can experience approximately 10 seconds.
- Cluster mode enabled may see approximately 1 second.

To modify a 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 **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 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 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 **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 Redis OSS cluster.

To add back nodes to your ElastiCache cluster, see Adding nodes to a cluster.

Managing clusters

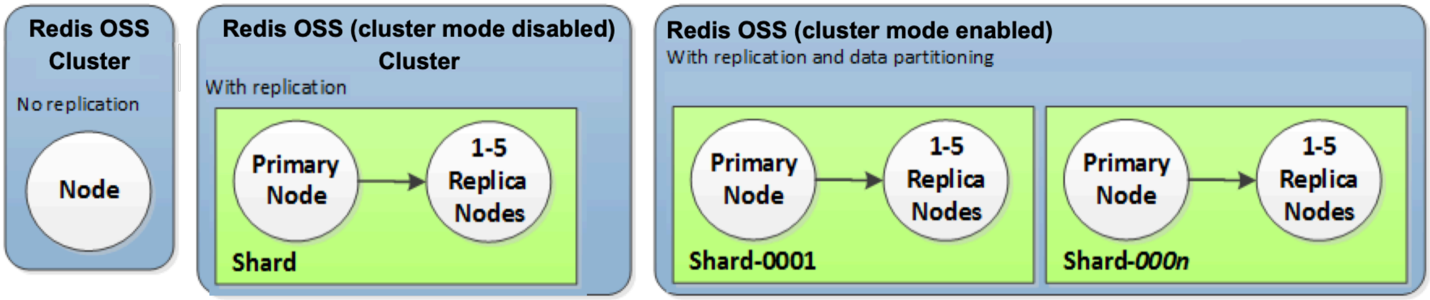
A *cluster* is a collection of one or more cache nodes, all of which run an instance of the Redis cache engine software. When you create a cluster, you specify the engine and version for all of the nodes to use.

The following diagram illustrates a typical Redis OSS cluster. Redis OSS clusters can contain a single node or up to six nodes inside a shard (API/CLI: node group), A single-node Redis OSS (cluster mode disabled) cluster has no shard, and a multi-node Redis OSS (cluster mode disabled) cluster has a single shard. 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 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**.

When you have multiple nodes in a shard, one of the nodes is a read/write primary node. All other nodes in the shard are read-only replicas.

Typical Redis OSS clusters look as follows.



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 Redis OSS versions, see [Supported ElastiCache (Redis OSS) versions](#).

Choosing a network type

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 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.

> **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.

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 (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.

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 (`cluster slots`, `cluster shards`, and `cluster nodes`) 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

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 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 Redis OSS (cluster mode enabled) cluster (Console) or Creating a 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 Redis OSS (Cluster Mode Disabled) replication group from scratch or Creating a replication group in Redis OSS (Cluster Mode Enabled) from scratch.

Using the CLI

When creating a cache cluster 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
```

Data tiering

Clusters that comprise a replication group and use a node type from the r6gd family have their data tiered between memory and local SSD (solid state drives) storage. Data tiering provides a new price-performance option for 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 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

Redis OSS commands and data structures supported in ElastiCache. You don't need any client-side changes to use this feature.

Topics

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 family, which is available in the following regions: `us-east-2`, `us-east-1`, `us-west-2`, `us-west-1`, `eu-west-1`, `eu-central-1`, `eu-north-1`, `eu-west-3`, `ap-northeast-1`, `ap-southeast-1`, `ap-southeast-2`, `ap-south-1`, `ca-central-1` and `sa-east-1`.
- You must use the Redis OSS 6.2 or later engine.
- You cannot restore a backup of an r6gd cluster into another cluster unless it also uses r6gd.
- You cannot export a backup to Amazon S3 for data-tiering clusters.
- Online migration is not supported for clusters running on the r6gd node type.

- Scaling is not supported from a data tiering cluster (for example, a cluster using an r6gd node type) to a cluster that does not use data tiering (for example, a cluster using an r6g node type). For more information, see Scaling ElastiCache (Redis OSS) .
- Auto scaling is supported on clusters using data tiering for Redis OSS version 7.0.7 and later. For more information, see Auto Scaling ElastiCache (Redis OSS) clusters
- Data tiering only supports `volatile-lru`, `allkeys-lru`, `volatile-lfu`, `allkeys-lfu` and `noeviction` maxmemory policies.
- Forkless save is supported for Redis OSS version 7.0.7 and later. For more information, see How synchronization and backup are implemented.
- Items larger than 128 MiB are not moved to SSD.

Pricing

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 (Redis OSS) 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 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 (Redis OSS) will start evicting items when the percentage of items in memory decreases below 5 percent. 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 the percentage of items in memory decreases below 5 percent. For more information on scaling see Scaling clusters in Redis OSS (Cluster Mode Enabled). For more information on metrics for Redis OSS clusters that use data tiering see Metrics for 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 family, such as *cache.r6gd.xlarge*. Selecting that node type automatically enables data tiering.

For more information on creating a cluster, see Creating a cluster.

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 family, such as *cache.r6gd.xlarge* and setting the `--data-tiering-enabled` parameter.

You cannot opt out of data tiering when selecting a node type from the r6gd family. If you set the `--no-data-tiering-enabled` parameter, the operation will fail.

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 node types in the r6gd family, 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 Redis OSS (cluster mode enabled) cluster.
    3. **Node Type –** Specify **cache.r6gd.xlarge** or any other node type from the r6gd family.
    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 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.

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 family, such as *cache.r6gd.xlarge* and setting the `--data-tiering-enabled` parameter.

You cannot opt out of data tiering when selecting a node type from the r6gd family. If you set the `--no-data-tiering-enabled` parameter, the operation will fail.

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

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 requirements

Preparation

Knowing the answers to the following questions helps make creating your 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 a parameter group.

  If you're running Redis OSS, consider setting `reserved-memory` or `reserved-memory-percent`. For more information, see Managing Reserved Memory.
- 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

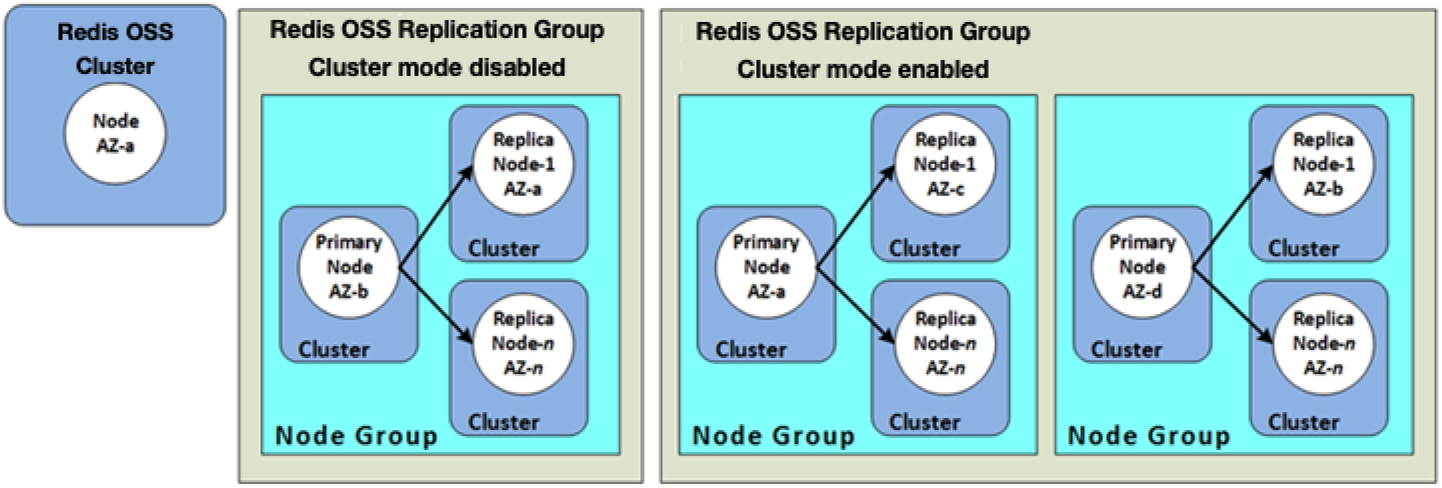
- Memory and processor requirements
- Redis OSS cluster configuration
- Scaling requirements
- Access requirements
- Region, Availability Zone and Local Zone requirements

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.

Redis OSS cluster configuration

ElastiCache (Redis OSS) clusters are comprised of from 0 to 500 shards (also called node groups). The data in a Redis OSS cluster is partitioned across the shards in the cluster. Your application connects with a Redis OSS cluster using a network address called an Endpoint. The nodes in a 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 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 (Redis OSS).



For more information, see Managing clusters.

Scaling requirements

All clusters can be scaled up by creating a new cluster with the new, larger node type. When you scale up a Redis OSS cluster, you can seed it from a backup and avoid having the new cluster start out empty.

For more information, see Scaling ElastiCache (Redis OSS) in this guide.

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

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
- Using local zones with ElastiCache
- Mitigating Failures

Choosing your node size

The node size you select for your cluster impacts costs, performance, and fault tolerance.

Choosing your node size

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 Redis OSS implementation:

- Do you expect throughput-bound workloads with multiple client connections?

  If this is the case and you're running Redis OSS version 5.0.6 or higher, you can get better throughput and latency with our enhanced I/O feature, where available CPUs are used for offloading the client connections, on behalf of the Redis OSS engine. If you're running Redis OSS version 7.0.4 or higher, on top of enhanced I/O, you will get additional acceleration with enhanced I/O multiplexing, where each dedicated network IO thread pipelines commands from multiple clients into the Redis OSS engine, taking advantage of Redis' ability to efficiently process commands in batches. In ElastiCache for Redis OSS v7.1 and above, we extended the enhanced I/O threads functionality to also handle the presentation layer logic. By presentation layer, what we mean is that Enhanced I/O threads are now not only reading client input, but also parsing the input into Redis OSS binary command format, which is then forwarded to the main thread for execution, providing performance gain. Refer to the blog post and the supported versions page for additional details.
- Do you have workloads that access a small percentage of their data regularly?

  If this is the case and you are running on Redis OSS engine version 6.2 or later, you can leverage data tiering by choosing the r6gd node type. 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.

  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.
- What version of Redis OSS are you running?

  Redis OSS versions before 2.8.22 require you to reserve more memory for failover, snapshot, synchronizing, and promoting a replica to primary operations. This requirement occurs because you must have sufficient memory available for all writes that occur during the process.

  Redis OSS version 2.8.22 and later use a forkless save process that requires less available memory than the earlier process.

  For more information, see the following:
  - How synchronization and backup are implemented
  - Ensuring that you have enough memory to create a 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 that you have enough memory to create a Redis OSS snapshot.
- Will your implementation be a standalone Redis OSS (cluster mode disabled) cluster or a Redis OSS (cluster mode enabled) cluster with multiple shards?

  Redis OSS (cluster mode disabled) cluster

If you're implementing a 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.m3.xlarge` node with 13.3 GB of memory or a `cache.r3.large` node with 13.5 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.m3.2xlarge` with 27.9 GB of memory or a `cache.r3.xlarge` with 30.5 GB of memory.

Redis OSS (cluster mode enabled) with multiple shards

If you're implementing a 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.m3.large` node with 6.05 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.m3.xlarge` with 13.3 GB of memory or a `cache.r3.large` with 13.5 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 the Redis OSS is 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.

Creating a cluster

The following examples show how to create a Redis OSS cluster using the AWS Management Console, AWS CLI and ElastiCache API.

Creating a Redis OSS (cluster mode disabled) (Console)

ElastiCache supports replication when you use the Redis OSS engine. To monitor the latency between when data is written to a 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 Redis OSS cluster might be added to a replication group at a later time, this key is included in all 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 Redis OSS (cluster mode disabled) cluster, follow the steps at Creating a Redis OSS (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.

Creating a Redis OSS (cluster mode enabled) cluster (Console)

If you are running Redis OSS 3.2.4 or later, you can create a Redis OSS (cluster mode enabled) cluster. 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 Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled), see Supported ElastiCache (Redis OSS) versions.

To create a 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 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 (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 Redis OSS-specific parameters and Creating a 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 an r6gd node type, data-tiering is automatically enabled. For more information, see Data tiering.

e. For **Number of shards**, choose the number of shards (partitions/node groups) that you want for this Redis OSS (cluster mode enabled) cluster.

   For some versions of Redis OSS (cluster mode enabled), you can change the number of shards in your cluster dynamically:

   - **Redis OSS 3.2.10 and later** – If your cluster is running Redis OSS 3.2.10 or later versions, you can change the number of shards in your cluster dynamically. For more information, see Scaling clusters in Redis OSS (Cluster Mode Enabled).
   - **Other Redis OSS versions** – If your cluster is running a version of Redis OSS before version 3.2.10, there's another approach. To change the number of shards in your cluster in this case, create a new cluster with the new number of shards. For more information, see Restoring from a backup into a new cache.

f. For **Replicas per shard**, choose the number of read replica nodes that you want in each shard.

   The following restrictions exist for 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 Seeding a new self-designed 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.
      - 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.

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 Redis OSS engine version 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.
          - **Redis OSS AUTH Default User** – An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.

        - **Redis OSS AUTH** – An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.

> **Note**
>
> For Redis OSS versions between 3.2.6 onward, excluding version 3.2.10, Redis OSS 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 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 (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.

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.

Creating a Redis OSS (cluster mode disabled) cluster (CLI)

Example – A Redis OSS (cluster mode disabled) Cluster with no read replicas

The following CLI code creates a Redis OSS (cluster mode disabled) cache cluster with no replicas.

> **Note**
>
> When creating cluster using a node type from the r6gd family, 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:::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 ^
--cache-parameter-group default.redis6.x ^
```

```
--snapshot-arns arn:aws:s3:::my_bucket/snapshot.rdb
```

Creating a Redis OSS (cluster mode enabled) cluster (AWS CLI)

Redis OSS (cluster mode enabled) clusters (API/CLI: replication groups) cannot be created using the `create-cache-cluster` operation. To create a Redis OSS (cluster mode enabled) cluster (API/CLI: replication group), see Creating a 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 (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.

Topics

Creating a Redis OSS (cluster mode disabled) cache cluster (ElastiCache API)

The following code creates a Redis OSS (cluster mode disabled) cache 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.redis3.2
    &Engine=redis
    &EngineVersion=3.2.4
```

```
&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 cache cluster in Redis OSS (cluster mode enabled) (ElastiCache API)

Redis OSS (cluster mode enabled) clusters (API/CLI: replication groups) cannot be created using the `CreateCacheCluster` operation. To create a Redis OSS (cluster mode enabled) cluster (API/CLI: replication group), see Creating a replication group in Redis OSS (Cluster Mode Enabled) from scratch (ElastiCache API).

For more information, see the ElastiCache API reference topic `CreateReplicationGroup`.

Viewing a cluster's details

You can view detail information about one or more clusters using the ElastiCache console, AWS CLI, or ElastiCache API.

Viewing details of a Redis OSS (Cluster Mode Disabled) cluster (Console)

You can view the details of a 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 Redis OSS (cluster mode disabled) cluster using the ElastiCache console.

To view a 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 console dashboard, choose **Redis OSS** to display a list of all your clusters that are running any version of Redis OSS.
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 Redis OSS engine, not 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 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 Redis OSS (Cluster Mode Enabled) cluster (Console)

You can view the details of a 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 Redis OSS (cluster mode enabled) cluster using the ElastiCache console.

To view a 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 console dashboard, choose **Redis OSS** to display a list of all your clusters that are running any version of Redis OSS.
4. To see details of a 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 Clustered Redis OSS engine, not just Redis OSS.

   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 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 a 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.redis3.2"
            },
            "SnapshotRetentionLimit": 0,
            "AutoMinorVersionUpgrade": true,
            "EngineVersion": "3.2.10",
            "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.redis3.2"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "3.2.10",
            "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.redis3.2.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "3.2.6",
            "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.redis3.2.cluster.on"
            },
            "SnapshotRetentionLimit": 0,
            "EngineVersion": "3.2.6",
            "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 a 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 a cluster, there can be times where you need to make other changes to an existing cluster, 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 Redis OSS-specific parameters.

Using the 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 Redis OSS clients to connect using both cluster mode enabled and cluster mode disabled. After you migrate all 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 Engine versions and upgrading . 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 node type from the r6gd family, you can only choose a different node size from within that family. If you choose a node type from the r6gd family, data tiering will automatically be 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

The **Apply Immediately** box applies only to engine version modifications. To apply changes immediately, choose the **Apply Immediately** check box. If this box is not chosen, node type and engine version modifications are applied during the next maintenance window. Other modifications, such as changing the maintenance window, are applied immediately.

7. Choose **Modify**.

To enable/disable log delivery

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/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**, 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.
   d. Choose **Enable**.

To change your configuration:

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

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 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 [Engine versions and upgrading](). 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 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 Engine versions and upgrading . 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 a cluster

To reconfigure your Redis OSS (cluster mode enabled) cluster, see Scaling clusters in Redis OSS (Cluster Mode Enabled)

You can use the ElastiCache Management Console, the AWS CLI or ElastiCache API to add nodes to your cluster.

Using the AWS Management Console

If you want to add a node to a single-node 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

The following procedure adds replication to a single-node 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 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 **Redis OSS clusters**.

   A list of clusters running the Redis OSS 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 a 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 a 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 Redis OSS (cluster mode enabled) cluster, see Scaling clusters in Redis OSS (Cluster Mode Enabled).

   If your cluster is a Redis OSS (cluster mode disabled) cluster with zero shards, first complete the steps at To add replication to a 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.

Scenarios	Pending Operation	New Request	Results
			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. 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. 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. 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. **Important** 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

If you want to add nodes to an existing 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 Redis OSS cache 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 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 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`.

Removing nodes from a cluster

You can delete a node from a cluster using the AWS Management Console, the AWS CLI, or the ElastiCache API.

Using the 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**
   >
   > If deleting the node results in the cluster no longer being 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. 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. 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. 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.
Scenario 4	Create	Create	The new create request is added to the pending create request. 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.

Scenarios	Pending Operation	New Request	Results
			Important 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

1. Identify the IDs of the nodes that you want to remove. For more information, see Viewing a 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

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`.

Canceling pending add or delete node operations

If you elected to not apply a change immediately, the operation has **pending** status until it is performed at your next maintenance window. You can cancel any pending operation.

To cancel a pending operation

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 you want to cancel a pending add or delete node operation in.
3. In the navigation pane, choose the engine running on the cluster that has pending operations that you want to cancel. A list of clusters running the chosen engine appears.
4. In the list of clusters, choose the name of the cluster, not the box to the left of the cluster's name, that has pending operations that you want to cancel.
5. 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.
6. Choose the **Nodes** tab.
7. To cancel all pending operations, click **Cancel Pending**. The **Cancel Pending** dialog box appears.
8. Confirm that you want to cancel all pending operations by choosing the **Cancel Pending** button, or to keep the operations, choose **Cancel**.

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 (Redis OSS) 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 console dashboard, choose the engine the cluster that you want to delete is running.

   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

The following code deletes the cache cluster `my-cluster`.

```
aws elasticache delete-cache-cluster --cache-cluster-id my-cluster
```

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

For more information, see the ElastiCache API reference topic [`DeleteCacheCluster`].

Accessing your 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 cluster or 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 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. The service is designed to be accessed exclusively from within AWS. However, if the ElastiCache cluster is hosted inside a VPC, you can use a Network Address Translation (NAT) instance to provide outside access.

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 NAT instance must be launched in the same VPC as the cluster.
- The NAT instance must be launched in a public subnet separate from the cluster.
- An Elastic IP Address (EIP) must be associated with the NAT instance. The port forwarding feature of iptables is used to forward a port on the NAT 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 cache 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 cache cluster. The impact increases with number of cache 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 information about how to set up high availability NAT on VPC, see High Availability for Amazon VPC NAT Instances: An Example.

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 Redis cluster with:
  - IP address – *10.0.1.230*
  - Default Redis port – *6379*
  - Security group – *sg-bd56b7da*
  - AWS instance IP address – *sg-bd56b7da*
- 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. Create a NAT instance in the same VPC as your cache cluster but in a public subnet.

   By default, the VPC wizard will launch a *cache.m1.small* node type. You should select a node size based on your needs. You must use EC2 NAT AMI to be able to access ElastiCache from outside AWS.

   For information about creating a NAT instance, see NAT Instances in the AWS VPC User Guide.

2. Create security group rules for the cache cluster and NAT instance.

   The NAT instance security group and the cluster instance should have the following rules:

   - Two inbound rules
     - One to allow TCP connections from trusted clients to each cache port forwarded from the NAT instance (6379 - 6381).
     - A second to allow SSH access to trusted clients.

     NAT instance security group - inbound rules

Type	Protocol	Port range	Source
Custom TCP Rule	TCP	6379-6380	198.51.100.27/32
SSH	TCP	22	203.0.113.73/32

   - 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 (Cluster instance Security Group)

   - 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-bd56b7da (Cluster 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. 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 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
```

5. 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
```

6. 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
- NAT Instances
- Configuring ElastiCache Clients
- High Availability for Amazon VPC NAT Instances: An Example

Finding connection endpoints

Your application connects to your cluster using endpoints. An endpoint is a node or cluster's unique address.

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.

Which endpoints to use

- **Redis OSS standalone node**, use the node's endpoint for both read and write operations.

- **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).

- **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 Redis OSS Cluster (Redis OSS 3.2). 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.

Finding a 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 a 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 (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 (Redis OSS) cluster's multiple read replicas in different AWS Availability Zones (AZ) to ensure high availability of reader endpoints.

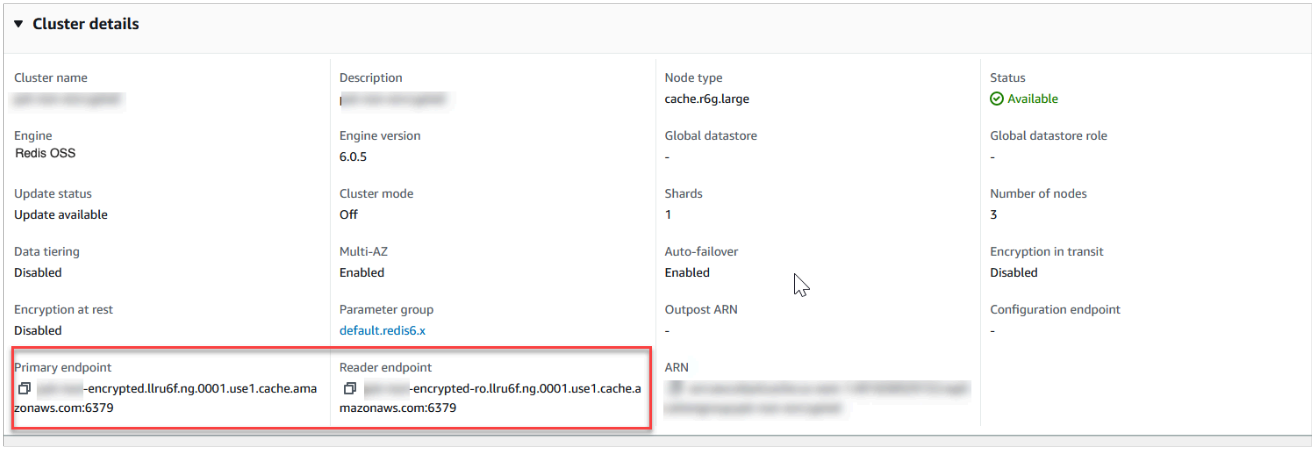
> **Note**
>
> A reader endpoint is not a load balancer. It is a DNS record that will resolve to an IP address of one of the replica nodes in a round robin fashion.

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 clusters**.

   The clusters screen will appear with a list of Redis OSS (cluster mode disabled) and 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).

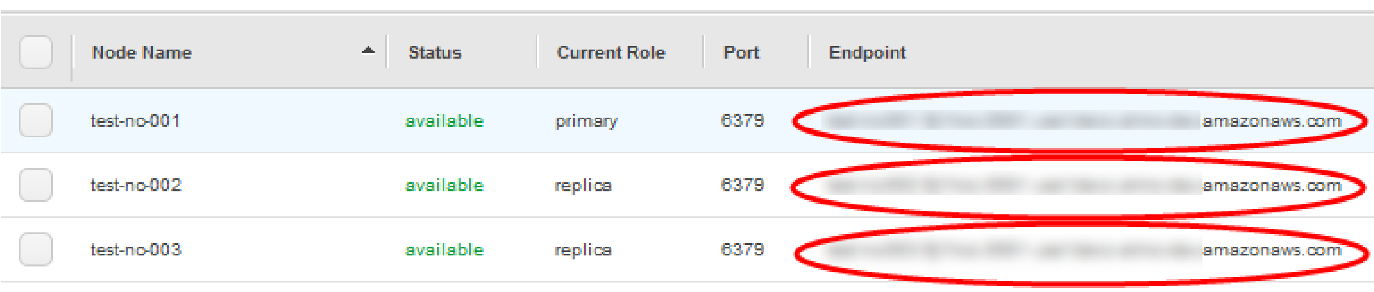


Primary and Reader endpoints for a Redis OSS (cluster mode disabled) cluster

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
```

Finding Endpoints for a Redis OSS (Cluster Mode Enabled) Cluster (Console)

A 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 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 **Redis OSS clusters**.

   The clusters screen will appear with a list of Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled) clusters. Choose the Redis OSS (cluster mode enabled) 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 Endpoints (AWS CLI)

You can use the AWS CLI for Amazon ElastiCache to discover the endpoints for nodes, clusters, and replication groups.

Topics

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 Redis clusters, the command returns the cluster 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 cluster information for the single-node Redis OSS (cluster mode disabled) cluster *mycluster*.

> **Important**
>
> The parameter `--cache-cluster-id` can be used with single-node Redis OSS (cluster mode disabled) cluster id or specific node ids in Redis replication groups. The `--cache-cluster-id` of a Redis replication group is a 4-digit value such as `0001`. If `--cache-cluster-id` is the id of a cluster (node) in a Redis 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.redis3.2"
            },
            "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": "3.2.10",
            "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 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)

You can use the Amazon ElastiCache API to discover the endpoints for nodes, clusters, and replication groups.

Topics

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 Redis clusters, the command returns the cluster endpoint. If you include the optional parameter `ShowCacheNodeInfo`, the action also returns the endpoints of the individual nodes in the cluster.

Example

```
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>
```

Finding Endpoints for 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

A shard (API/CLI: node group) is a collection of one to six Redis OSS nodes. A 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 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.

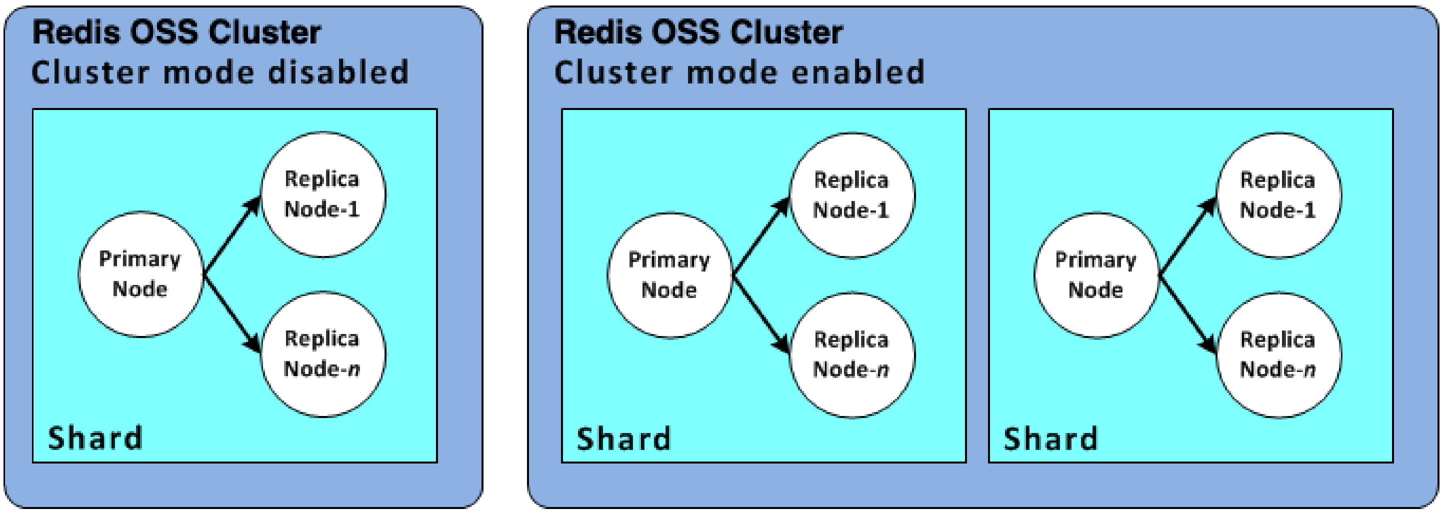
For versions below 5.0.6, the limit is 250 per cluster.

To request a limit increase, see AWS service limits and choose the limit type **Nodes per cluster per instance type**.

When you create a 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 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.



Redis OSS shard configurations

For more information, see Offline resharding and shard rebalancing for Redis OSS (cluster mode enabled) and Online resharding and shard rebalancing for 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 Redis OSS (Cluster Mode Disabled)
- For Redis OSS (Cluster Mode Enabled)

For Redis OSS (Cluster Mode Disabled)

Redis OSS (cluster mode disabled) replication group shard IDs are always `0001`.

For Redis OSS (Cluster Mode Enabled)

The following procedure uses the AWS Management Console to find a Redis OSS (cluster mode enabled)'s replication group's shard ID.

To find the shard ID in a 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 **Redis OSS**, then choose the name of the 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 Redis OSS (cluster mode disabled) 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.m3.medium",
            "DataTiering": "disabled",
            "PendingModifiedValues": {}
        }
    ]
}
```

Using the ElastiCache API

To find shard (node group) ids for either Redis OSS (cluster mode disabled) 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 Memcached and Redis OSS self-designed caches

Amazon ElastiCache supports the 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, self-designed 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, self-designed 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 Redis OSS with a version of ElastiCache (Redis OSS) if the following apply for you:

- **ElastiCache (Redis OSS) version 7.0 (Enhanced)**

You want to use Redis OSS Functions, Sharded Pub/Sub, or Redis OSS ACL improvements. For more information, see Redis OSS Version 7.0 (Enhanced).

- **ElastiCache (Redis OSS) version 6.2 (Enhanced)**

  You want the ability to tier data between memory and SSD using the r6gd node type. For more information, see Data tiering.
- **ElastiCache (Redis OSS) version 6.0 (Enhanced)**

  You want to authenticate users with role-based access control.

  For more information, see Redis OSS Version 6.0 (Enhanced).
- **ElastiCache (Redis OSS) version 5.0.0 (Enhanced)**

  You want to use Redis OSS streams, a log data structure that allows producers to append new items in real time and also allows consumers to consume messages either in a blocking or non-blocking fashion.

  For more information, see Redis OSS Version 5.0.0 (Enhanced).
- **ElastiCache (Redis OSS) version 4.0.10 (Enhanced)**

  Supports both encryption and dynamically adding or removing shards from your Redis OSS (cluster mode enabled) cluster.

  For more information, see Redis OSS Version 4.0.10 (Enhanced).

The following versions are deprecated, have reached or soon to reach end of life.

- **ElastiCache (Redis OSS) version 3.2.10 (Enhanced)**

  Supports the ability to dynamically add or remove shards from your Redis OSS (cluster mode enabled) cluster.

  > **Important**
  > Currently ElastiCache (Redis OSS) 3.2.10 doesn't support encryption.

  For more information, see the following:

  - Redis OSS Version 3.2.10 (Enhanced)
  - Online resharding best practices for Redis OSS, For more information, see the following:
    - Best Practices: Online Resharding
    - Online Resharding and Shard Rebalancing for Redis OSS (Cluster Mode Enabled)
  - For more information on scaling Redis OSS clusters, see Scaling.

- **ElastiCache (Redis OSS) version 3.2.6 (Enhanced)**

  If you need the functionality of earlier Redis OSS versions plus the following features, choose ElastiCache (Redis OSS) 3.2.6:

  - In-transit encryption. For more information, see Amazon ElastiCache (Redis OSS) In-Transit Encryption.
  - At-rest encryption. For more information, see Amazon ElastiCache (Redis OSS) At-Rest Encryption.

- **ElastiCache (Redis OSS) (Cluster mode enabled) version 3.2.4**

  If you need the functionality of Redis OSS 2.8.x plus the following features, choose Redis OSS 3.2.4 (clustered mode):

  - You need to partition your data across two to 500 node groups (clustered mode only).
  - You need geospatial indexing (clustered mode or non-clustered mode).
  - You don't need to support multiple databases.

- **ElastiCache (Redis OSS) (non-clustered mode) 2.8.x and 3.2.4 (Enhanced)**

  If the following apply for you, choose Redis OSS 2.8.x or Redis OSS 3.2.4 (non-clustered mode):

  - You need complex data types, such as strings, hashes, lists, sets, sorted sets, and bitmaps.
  - You need to sort or rank in-memory datasets.
  - You need persistence of your key store.
  - You need to replicate your data from the primary to one or more read replicas for read intensive applications.
  - You need automatic failover if your primary node fails.
  - You need publish and subscribe (pub/sub) capabilities—to inform clients about events on the server.
  - You need backup and restore capabilities for self-designed clusters as well as serverless caches.
  - You need to support multiple databases.

Comparison summary of Memcached, Redis OSS (cluster mode disabled), and Redis OSS (cluster mode enabled)

	Memcached	Redis OSS (cluster mode disabled)	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 * Complex	3.2.x and later - Complex
Data partitioning	Yes	No	Yes
Cluster is modifiable	Yes	Yes	3.2.10 and later - Limited
Online resharding	No	No	3.2.10 and later
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
Compliance certifications			
Compliance Certification			
FedRAMP	Yes - 1.6.12 and later	4.0.10 and later	4.0.10 and later
HIPAA	Yes - 1.6.12 and later	4.0.10 and later	4.0.10 and later
PCI DSS	Yes	4.0.10 and later	4.0.10 and later
Multi-threaded	Yes	No	No
Node type upgrade	No	Yes	Yes
Engine upgrading	Yes	Yes	Yes

	Memcached	Redis OSS (cluster mode disabled)	Redis OSS (cluster mode enabled)
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 Memcached only, not for self-designed Memcached clusters	Yes	Yes
Geospatial indexing	No	4.0.10 and later	Yes

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 ElastiCache (Memcached) Versions or Supported ElastiCache (Redis OSS) Versions.

Online migration to ElastiCache

By using Online Migration, you can migrate your data from self-hosted open-source Redis OSS on Amazon EC2 to Amazon ElastiCache.

Note

Online migration is not supported to ElastiCache serverless caches or clusters running on the r6gd node type.

Overview

To migrate your data from open-source 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 instance type, number of shards, and number of replicas.

Online migration is designed for data migration from self-hosted open-source Redis OSS on Amazon EC2 to ElastiCache for Redis OSS and not between ElastiCache for Redis OSS 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. For Redis OSS cluster-mode disabled, the migration process makes the primary node of the ElastiCache for Redis OSS cluster a replica of your source Redis OSS primary. For Redis OSS cluster-mode enabled, 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 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 redis-cli INFO command on your Redis OSS nodes and on the ElastiCache primary nodes.

Migration steps

The following topics outline the process for migrating your data:

Preparing your source and target Redis OSS nodes for migration

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 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 Redis OSS engine version 5.0.6 or higher.
   - It doesn't have either encryption in-transit or encryption at-rest enabled.
   - It has Multi-AZ enabled.
   - It has sufficient memory available to fit the data from your Redis OSS cluster. To configure the right reserved memory settings, see Managing Reserved Memory.
   - For cluster-mode disabled, you can migrate directly from Redis OSS versions 2.8.21 onward to Redis OSS version 5.0.6 onward if are using the CLI or Redis OSS versions 5.0.6 onward using the CLI or console. For cluster mode enabled, you can migrate directly from any cluster-mode enabled Redis OSS version to Redis OSS version 5.0.6 onward if are using the CLI or Redis OSS versions 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 open-source Redis OSS and the ElastiCache (Redis OSS) deployment are compatible.

   At a minimum, all the following in the target ElastiCache deployment should be compatible with your Redis OSS configuration for Redis OSS replication:

- Your Redis OSS cluster should not have Redis OSS AUTH enabled.
- Redis OSS config `protected-mode` should be set to `no`.
- If you have `bind` configuration in your 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 Redis OSS cluster. This value is set using `databases` in the Redis OSS config.
- 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 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 Redis OSS cluster and transferred over the network to ElastiCache node.
- All 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 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 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 migration setup 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 data migration 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 recommended to use `TestMigration` API to validate migration setup. But this is strictly optional.

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 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 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 Redis OSS instance (in corresponding shard owning same slots in cluster enabled redis). The status of ElastiCache cluster changes to **migrating** and data starts migrating from your Redis OSS instance to the ElastiCache primary node. Depending on the size of the data and load on your Redis OSS instance, the migration can take a while to complete. You can check the progress of the migration by running the redis-cli INFO command on your Redis OSS instance and ElastiCache primary node.

After successful replication, all writes to your 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 Redis OSS cluster gets replicated to all the nodes in ElastiCache cluster.

If an ElastiCache primary node can't become a replica of your 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 Redis OSS nodes for migration.
- Ensure that the routing configuration for your VPC and subnets allows traffic between ElastiCache nodes and your Redis OSS instances.
- Ensure the security group attached to your Redis OSS instances allows input bound traffic from ElastiCache nodes.
- Check Redis OSS logs for your Redis OSS instances for more information about failures specific to replication.

Verifying the data migration progress

After the data migration has begun, you can do the following to track its progress:

- Verify that 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 Redis OSS instances. Doing this also provides information about replication lag.
- Verify low client output buffer by using the [CLIENT LIST](https://redis.io/commands/client-list) Redis OSS command on your 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 Redis OSS cluster.

Completing the data migration

When you are ready to cut over to the 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 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 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 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 the complete the migration step after the replacement is complete.

Performing online data migration using the Console

You can use the AWS Management Console to migrate your data from your cluster to your Redis OSS 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 Redis OSS cluster or choose an existing cluster. Make sure that the cluster meets the following requirements:
   - Your Redis OSS engine version should be 5.0.6 or higher.
   - Your Redis OSS cluster should not have Redis OSS AUTH enabled.
   - Redis OSS config `protected-mode` should be set to no.
   - If you have `bind` configuration in your Redis OSS config, then it should be updated to allow requests from ElastiCache nodes.
   - The number of databases should be the same between the ElastiCache node and your Redis OSS cluster. This value is set using `databases` in the Redis OSS config.
   - Redis OSS commands that perform data modification should not be renamed to allow replication of the data to succeed.
   - To replicate the data from your 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 Redis OSS cluster and transferred over the network to ElastiCache node.
   - The cluster is in **available** status.
3. With your cluster selected, choose **Migrate Data from Endpoint** for **Actions**.
4. In the **Migrate Data from Endpoint** dialog box, enter the IP address, and the port where your Redis OSS cluster is available.

   > **Important**
   >
   > The IP address must be exact. If you enter the address incorrectly, the migration fails.

5. Choose **Start Migration**.

   As the cluster begins migration, it changes to **Modifying** and then **Migrating** status.

6. Monitor the migration progress by choosing **Events** on the navigation pane.

At any point during the migration process, you can stop migration. To do so, choose your cluster and choose **Stop Data Migration** for **Actions**. The cluster then goes to **Available** status.

If the migration succeeds, the cluster goes to **Available** status and the event log shows the following:

```
Migration operation succeeded for replication group ElastiCacheClusterName.
```

If the migration fails, the cluster goes to **Available** status and the event log shows the following:

```
Migration operation failed for replication group ElastiCacheClusterName.
```

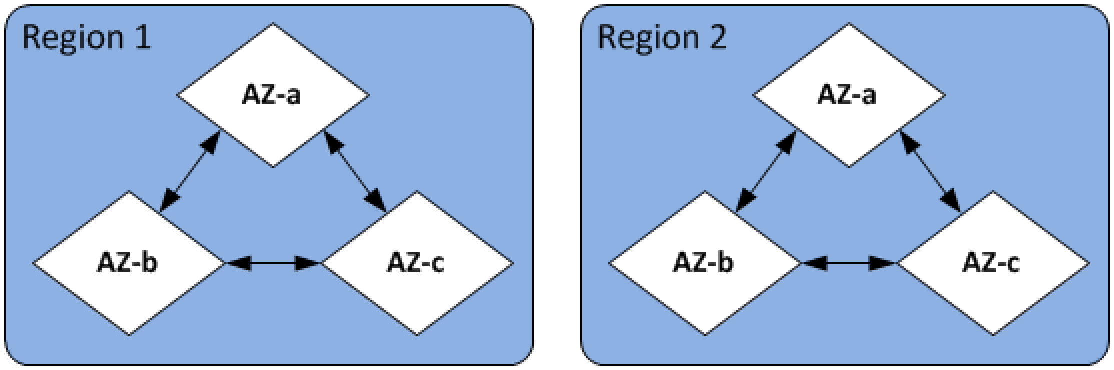
Choosing regions and availability zones

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

- Locating your nodes
- Supported regions & endpoints
- Using local zones with ElastiCache
- Using Outposts

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. For more information, see the following:

- Creating a cluster
- Modifying an ElastiCache cluster
- Adding nodes to a 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 at the top of this topic.

Regions where ElastiCache is supported

Region Name/Region	Endpoint	Protocol	
US East (Ohio) Region `us-east-2`	`elasticache.us-east-2.amazonaws.com`	HTTPS	
US East (N. Virginia) Region `us-east-1`	`elasticache.us-east-1.amazonaws.com`	HTTPS	
US West (N. California) Region `us-west-1`	`elasticache.us-west-1.amazonaws.com`	HTTPS	
US West (Oregon) Region `us-west-2`	`elasticache.us-west-2.amazonaws.com`	HTTPS	
Canada (Central) Region `ca-central-1`	`elasticache.ca-central-1.amazonaws.com`	HTTPS	
Canada (West) Region `ca-west-1`	`elasticache.ca-west-1.amazonaws.com`	HTTPS	
Asia Pacific (Jakarta) `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 ap-south-2	elasticache.ap-south-2.amazonaws.com	HTTPS
Asia Pacific (Tokyo) Region ap-northeast-1	elasticache.ap-northeast-1.amazonaws.com	HTTPS
Asia Pacific (Seoul) Region ap-northeast-2	elasticache.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Osaka) Region ap-northeast-3	elasticache.ap-northeast-3.amazonaws.com	HTTPS
Asia Pacific (Singapore) Region ap-southeast-1	elasticache.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney) Region ap-southeast-2	elasticache.ap-southeast-2.amazonaws.com	HTTPS
Europe (Frankfurt) Region eu-central-1	elasticache.eu-central-1.amazonaws.com	HTTPS

Region Name/Region	Endpoint	Protocol
Europe (Zurich) Region `eu-central-2`	`elasticache.eu-central-2.amazonaws.com`	HTTPS
Europe (Stockholm) Region `eu-north-1`	`elasticache.eu-north-1.amazonaws.com`	HTTPS
Middle East (Bahrain) Region `me-south-1`	`elasticache.me-south-1.amazonaws.com`	HTTPS
Middle East (UAE) Region `me-central-1`	`elasticache.me-central-1.amazonaws.com`	HTTPS
Europe (Ireland) Region `eu-west-1`	`elasticache.eu-west-1.amazonaws.com`	HTTPS
Europe (London) Region `eu-west-2`	`elasticache.eu-west-2.amazonaws.com`	HTTPS
EU (Paris) Region `eu-west-3`	`elasticache.eu-west-3.amazonaws.com`	HTTPS

Region Name/Region	Endpoint	Protocol
Europe (Milan) Region `eu-south-1`	`elasticache.eu-south-1.amazonaws.com`	HTTPS
Europe (Spain) Region `eu-south-2`	`elasticache.eu-south-2.amazonaws.com`	HTTPS
South America (São Paulo) Region `sa-east-1`	`elasticache.sa-east-1.amazonaws.com`	HTTPS
China (Beijing) Region `cn-north-1`	`elasticache.cn-north-1.amazonaws.com.cn`	HTTPS
China (Ningxia) Region `cn-northwest-1`	`elasticache.cn-northwest-1.amazonaws.com.cn`	HTTPS
Asia Pacific (Hong Kong) Region `ap-east-1`	`elasticache.ap-east-1.amazonaws.com`	HTTPS
Africa (Cape Town) Region `af-south-1`	`elasticache.af-south-1.amazonaws.com`	HTTPS

Region Name/Region	Endpoint	Protocol
Israel (Tel Aviv) Region `il-central-1`	`elasticache.il-central-1.amazonaws.com`	HTTPS
AWS GovCloud (US-West) `us-gov-west-1`	`elasticache.us-gov-west-1.amazonaws.com`	HTTPS
AWS GovCloud (US-East) `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.

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 AWS 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 in the *ElastiCache User Guide*.
4. Create an ElastiCache (Redis OSS) cluster that uses the ElastiCache subnet in the Local Zone. For more information, see one of the following topics:

- [Creating a Redis OSS (cluster mode disabled) cluster (Console)]
- [Creating a Redis OSS (cluster mode enabled) cluster (Console)]

Using Outposts

AWS 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 Redis OSS 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 **Redis OSS caches**.
3. Select **Create Redis OSS 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 cache cluster or, if there are no available Outposts, create a new one using the following steps:

Under On-Premises options:

1. Under **Redis OSS settings**:

   a. **Name**: Enter a name for the Redis OSS cluster
   b. **Description**: Enter a description for the Redis OSS cluster.
   c. **Engine version compatilbility**: Engine version is based on the AWS Outpost region
   d. **Port**: Accept the default port, 6379. If you have a reason 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. 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 Redis OSS 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 cache cluster that uses Outposts:

- [create-cache-cluster](#) – Using this operation, the `outpost-mode` parameter accepts a value that specifies whether the nodes in the cache 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 cache 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

Amazon ElastiCache caches running Redis OSS 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 Redis OSS 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

Consider the following constraints when planning or making backups:

- Backup and restore are supported only for caches running on Redis OSS or Serverless Memcached.
- For Redis OSS (cluster mode disabled) clusters, backup and restore aren't supported on `cache.t1.micro` nodes. All other cache node types are supported.
- For 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 Redis OSS self-designed clusters, you can create no more than 20 manual backups per node in the cluster.
- Redis OSS (cluster mode enabled) only supports taking backups on the cluster level (for the API or CLI, the replication group level). 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 self-designed cluster during backup.
- If using caches with data tiering, you cannot export a backup to Amazon S3.
- You can restore a backup of a cluster using the r6gd node type only to clusters using the r6gd node type.

Performance impact of backups of self-designed clusters

Backups on serverless caches are transparent to the application with no performance impact. However, when creating backups for self-designing clusters, there can be some performance impact depending on the available reserved memory. Self-designed clusters are not available with ElastiCache and Memcached, but are available with ElastiCache and Redis OSS.

The following are guidelines for improving backup performance for self-designed clusters.

- Set the `reserved-memory-percent` parameter – To mitigate excessive paging, we recommend that you set the *reserved-memory-percent* parameter. This parameter prevents 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.

- Create backups from a read replica – If you are running 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 Redis OSS data, before the replication group is deleted.

Scheduling automatic backups

You can enable automatic backups for any Redis OSS serverless cache or self-designed 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.

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.

You can enable or disable automatic backups when either creating a new cache or updating an existing Redis OSS cache, by using the ElastiCache console, the AWS CLI, or the ElastiCache API. This is done by checking the **Enable Automatic Backups** box in the **Advanced Redis OSS 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.

Creating a manual backup (Console)

To create a backup of a cache (console)

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation pane, choose **Redis OSS caches**.
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-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-snapshot \
                          --serverless-cache-name CacheName \
                          --serverless-cache-snapshot-name bkup-20231127
  ```

For Windows:

- ```
  aws elasticache create-serverless-snapshot ^
      --serverless-cache-name CacheName ^
      --serverless-cache-snapshot-name bkup-20231127
  ```

Manual backup of a self-designed cluster with the AWS CLI

To create a manual backup of a self-designed 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 Redis OSS (cluster mode disabled) cluster.

- `--replication-group-id` – Name of the 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 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 Redis OSS (Cluster Mode Disabled) cluster that has no replica nodes

The following AWS CLI operation creates the backup `bkup-20150515` from the 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 Redis OSS (Cluster Mode Disabled) cluster with replica nodes

The following AWS CLI operation creates the backup `bkup-20150515` from the 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 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.m3.2xlarge",
        "DataTiering": "disabled"
    }
}
```

Example 3: Backing up a cluster for Redis OSS (Cluster Mode Enabled)

The following AWS CLI operation creates the backup `bkup-20150515` from the Redis OSS (cluster mode enabled) cluster `myClusteredRedis`. Note the use of `--replication-group-id` instead of `--cache-cluster-id` to identify the source.

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 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.r3.large",
        "DataTiering": "disabled"
    }
}
```

Related topics

For more information, see create-snapshot in the *AWS CLI Command Reference*.

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 Redis OSS serverless cache or self-designed 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 serverless cache
- When deleting a Redis OSS self-designed cluster with no read replicas
- When deleting a Redis OSS cluster with read replicas

When deleting a serverless 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 Redis OSS self-designed cluster with no read replicas

To create a final backup for a self-designed 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 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 self-designed cluster backups (AWS CLI)

To display a list of self-designed 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 self designed cluster backup (AWS CLI)

To copy a backup of a self-designed 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. Available for Redis OSS serverless caches and Redis OSS self-designed clusters only. 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 my-backup-copy
```

For Windows:

```
aws elasticache copy-snapshot ^
    --source-snapshot-name automatic.my-redis-primary-2014-03-27-03-15 ^
    --target-snapshot-name my-backup-copy
```

For more information, see copy-snapshot in the *AWS CLI*.

Exporting a backup

Amazon ElastiCache supports exporting your ElastiCache (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 Seeding a new self-designed 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.
- Exporting a backup is available for Redis OSS self-designed clusters, Serverless Redis OSS and Serverless Memcached. Exporting a backup is not available for self-designed 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.

Step 1: 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*.

Step 2: Grant ElastiCache access to your Amazon S3 bucket

For ElastiCache to be able to copy a snapshot to an Amazon S3 bucket, you must update your bucket policy to grant ElastiCache access to the bucket.

> **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*.

To create the proper permissions on an Amazon S3 bucket, take the steps described following.

To grant ElastiCache access to an 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 Amazon S3 bucket that you want to copy the backup to. This should be the S3 bucket that you created in Step 1: Create an Amazon S3 bucket.
3. Choose the **Permissions** tab and under **Permissions**, choose **Access control list (ACL)** and then choose **Edit**.
4. Add grantee Canonical Id `540804c33a284a299d2547575ce1010f2312ef3da9b3a053c8bc45bf233e4353` with the following options:

   - **Objects: List, Write**
   - **Bucket ACL: Read, Write**

   > **Note**
   >
   > - For the PDT GovCloud Region, the Canonical Id is `40fa568277ad703bd160f66ae4f83fc9dfdfd06c2f1b5060ca22442ac3ef8be6`.
   > - For the OSU GovCloud Region, the Canonical Id is `c54286759d2a83da9c480405349819c993557275cf37d820d514b42da6893f5c`.

5. Choose **Save**.

Step 3: 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. The following examples assume that the IAM identity of the caller has the following additional S3 specific IAM permissions.

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

```
      "s3:GetBucketLocation",
      "s3:ListAllMyBuckets",
      "s3:PutObject",
      "s3:GetObject",
      "s3:DeleteObject",
      "s3:ListBucket"
    ],
    "Resource": "arn:aws:s3:::*"
  }]
}
```

For opt-in Regions, the following is an example of what the updated policy for the S3 bucket might look like. (This examples uses the Asia Pacific (Hong Kong) Region.)

```
{
    "Version": "2012-10-17",
    "Id": "Policy15397346",
    "Statement": [
        {
            "Sid": "Stmt15399483",
            "Effect": "Allow",
            "Principal": {
                "Service": "elasticache.amazonaws.com"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::hkg-elasticache-backup",
                "arn:aws:s3:::hkg-elasticache-backup/*"
            ]
        },
        {
            "Sid": "Stmt15399484",
            "Effect": "Allow",

            "Principal": {
                "Service": "ap-east-1.elasticache-snapshot.amazonaws.com"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::hkg-elasticache-backup",
                "arn:aws:s3:::hkg-elasticache-backup/*"
            ]
        }
```

```
    ]
}
```

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 **Copy**.
5. In **Create a Copy of the 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 Step 1: 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 Step 2: 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 Step 2: 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 self-designed ElastiCache cluster backup (AWS CLI)

Exporting a backup of a self-designed 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 Step 2: 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 into a new serverless cache or a self-designed cluster.

Restoring a backup into a serverless cache (Console)

> **Note**
>
> ElastiCache Serverless supports RDB files compatible with 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 self-designed cluster (Console)

To restore a backup to a self-designed 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 **Design your own 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 self-designed cache and import data from your backup.

Restoring a backup into a serverless cache (AWS CLI)

> **Note**
>
> ElastiCache Serverless supports RDB files compatible with 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
```

For Windows:

Restoring a backup into a self-designed cluster (AWS CLI)

To restore a backup to a self-designed cluster (AWS CLI)

You can restore a Redis OSS serverless cache backup, and you can also resotore a Redis OSS self-designed cluster.

You can restore a Redis OSS serverless cache backup in two ways.

- You can restore to a single-node Redis OSS (cluster mode disabled) cluster using the AWS CLI operation create-cache-cluster.
- You can restore to a Redis OSS cluster with read replicas (a replication group). To do this, you can use either Redis OSS (cluster mode disabled) or Redis OSS (cluster mode enabled) with

the AWS CLI operation create-replication-group. In this case, you seed the restore with a Redis OSS .rdb file. For more information on seeding a new self-designed cluster, see Seeding a new self-designed cluster with an externally created backup.

You can restore a Redis OSS (cluster mode disabled) backup in two ways.

- You can restore to a single-node Redis OSS (cluster mode disabled) cluster using the AWS CLI operation create-cache-cluster.
- You can restore to a Redis OSS cluster with read replicas (a replication group). To do this, you can use either Redis OSS (cluster mode disabled) or Redis OSS (cluster mode enabled) with the AWS CLI operation create-replication-group. In this case, you seed the restore with a Redis OSS .rdb file. For more information on seeding a new self-designed cluster, see Seeding a new self-designed cluster with an externally created backup.

When using either the `create-cache-cluster` or `create-replication-group` operation, be sure to include the parameter `--snapshot-name` or `--snapshot-arns` to seed the new cluster or replication group with the data from the 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 self-designed cluster backup (AWS CLI)

Use the delete-snapshot AWS CLI operation with the following parameter to delete a self-designed 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.

Seeding a new self-designed cluster with an externally created backup

When you create a new Redis OSS self-designed cluster, you can seed it with data from a Redis OSS .rdb backup file. Seeding the cluster is useful if you currently manage a Redis OSS instance outside of ElastiCache and want to populate your new ElastiCache for Redis OSS self-designed cluster with your existing Redis OSS data.

To seed a new Redis OSS self-designed cluster from a Redis OSS backup created within Amazon ElastiCache, see Restoring from a backup into a new cache.

When you use a Redis OSS .rdb file to seed a new Redis OSS self-designed cluster, you can do the following:

- Upgrade from a nonpartitioned cluster to a Redis OSS (cluster mode enabled) self-designed cluster running Redis OSS version 3.2.4.
- Specify a number of shards (called node groups in the API and CLI) in the new self-designed cluster. This number can be different from the number of shards in the self-designed cluster that was used to create the backup file.
- Specify a different node type for the new self-designed 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 Redis OSS overhead. For more information, see Ensuring that you have enough memory to create a Redis OSS snapshot.
- Distribute your keys in the slots of the new 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 Redis OSS (cluster mode disabled) cluster from an .rdb file created from a Redis OSS (cluster mode enabled) cluster.

Important

- You must ensure that your 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 Redis OSS data to a cache.m3.medium node that has 2.9 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 Redis OSS .rdb file with Amazon S3 server-side encryption (SSE-S3) only. For more information, see Protecting data using server-side encryption.

Following, you can find topics that walk you through migrating your Redis OSS cluster from outside ElastiCache for Redis OSS to ElastiCache for Redis OSS.

Migrating to ElastiCache for Redis OSS

Step 1: Create a Redis OSS backup

To create the Redis OSS backup to seed your ElastiCache for Redis OSS instance

1. Connect to your existing Redis OSS instance.
2. Run either the Redis OSS 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 Redis OSS website.

   SAVE is synchronous and blocks other processes until finished. For more information, see SAVE at the Redis OSS website.

For additional information on creating a backup, see Redis OSS persistence at the Redis OSS 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 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 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 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 **Permissions**.
6. If **aws-scs-s3-readonly** or one of the canonical IDs in the following list is not listed as a user, do the following:
   a. Under **Access for other AWS accounts**, choose **Add grantee**.
   b. In the box, add the AWS Region's canonical ID as shown following:
      - AWS GovCloud (US-West) Region:

        ```
        40fa568277ad703bd160f66ae4f83fc9dfdfd06c2f1b5060ca22442ac3ef8be6
        ```

        > **Important**
        >
        > The backup must be located in an S3 bucket in AWS GovCloud (US) for you to download it to a Redis OSS cluster in AWS GovCloud (US).

      - AWS Regions enabled by default:

        ```
        540804c33a284a299d2547575ce1010f2312ef3da9b3a053c8bc45bf233e4353
        ```
   c. Set the permissions on the bucket by choosing **Yes** for the following:
      - **List/write object**
      - **Read/write object ACL permissions**
   d. Choose **Save**.
7. Choose **Overview**, and then choose **Download**.

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.

```
{
    "Version": "2012-10-17",
    "Id": "Policy15397346",
    "Statement": [
        {
            "Sid": "Stmt15399483",
            "Effect": "Allow",
            "Principal": {
                "Service": "ap-east-1.elasticache-snapshot.amazonaws.com"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketAcl"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket",
```

```
                "arn:aws:s3:::example-bucket/backup1.rdb",
                "arn:aws:s3:::example-bucket/backup2.rdb"
            ]
        }
    ]
}
```

8. Choose **Save changes**.

Step 5: 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 or Creating a Redis OSS replication group from scratch. Be sure to choose Redis OSS as your cluster engine.

The method you use to tell ElastiCache where to find the Redis OSS backup you uploaded to Amazon S3 depends on the method you use to create the cluster:

Seed the ElastiCache (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 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 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

This section covers the supported Redis OSS engine versions and how to upgrade.

Topics

Engine versions and upgrading

ElastiCache (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 self-designed clusters, ElastiCache (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.

Version management for ElastiCache Serverless

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.

Version management for self-designed ElastiCache clusters

When working with self-designed ElastiCache clusters, you can control when the software powering your cache 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 cache 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.

You initiate engine version upgrades to your cluster or replication group by modifying it and specifying a new engine version. For more information, see the following:

- Modifying clusters
- Modifying a replication group

Upgrade considerations when working with self-designed clusters

> **Note**
>
> The following considerations only apply when upgrading self-designed clusters. They do not apply to ElastiCache Serverless.

When upgrading a self-designed cluster, consider the follwing

- 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 Redis OSS 6.0, ElastiCache (Redis OSS) will offer a single version for each Redis OSS 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 (Redis OSS) supports switching between Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled).
- The Amazon ElastiCache (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 (Redis OSS) 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 Redis OSS (cluster mode disabled) to Redis OSS (cluster mode enabled) when you upgrade your engine. The following procedure shows you how to upgrade from Redis OSS (cluster mode disabled) to Redis OSS (cluster mode enabled).

To upgrade from a Redis OSS (cluster mode disabled) to Redis OSS (cluster mode enabled) engine version

1. Make a backup of your 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 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 Seeding a new self-designed cluster with an externally created backup.
3. Delete the old Redis OSS (cluster mode disabled) cluster or replication group. For more information, see Deleting a cluster or Deleting a replication group.
4. Scale the new Redis OSS (cluster mode enabled) cluster or replication group to the number of shards (node groups) that you need. For more information, see Scaling clusters in Redis OSS (Cluster Mode Enabled)

- 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 that you have enough memory to create a 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, all replicas are processed before the primary is processed. If there are fewer replicas in a shard, the primary in that shard might be processed before the replicas in other shards are finished processing.
  - Across all the shards, primary nodes are processed in series. Only one primary node is upgraded 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, such as from 3.2.6 to 3.2.10.

How to upgrade engine versions

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. For more information, see the following topics.

How to modify clusters and replication groups

Clusters	Replication groups
Using the AWS Management Console	Using the AWS Management Console
Using the AWS CLI	Using the AWS CLI
Using the ElastiCache API	Using the ElastiCache API

Resolving blocked Redis OSS engine upgrades

As shown in the following table, your 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 Redis OSS engine upgrade

- Do one of the following:

- Schedule your Redis OSS 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.

Supported ElastiCache (Redis OSS) versions

ElastiCache Serverless caches support the following Redis OSS versions:

- ElastiCache (Redis OSS) version 7.1 (enhanced)

Self-designed ElastiCache clusters support the following Redis OSS versions:

- ElastiCache (Redis OSS) version 7.1 (enhanced)
- ElastiCache (Redis OSS) version 7.0 (enhanced)
- ElastiCache (Redis OSS) version 6.2 (enhanced)
- ElastiCache (Redis OSS) version 6.0 (enhanced)
- ElastiCache (Redis OSS) version 5.0.6 (enhanced)
- ElastiCache (Redis OSS) version 5.0.5 (deprecated, use version 5.0.6)
- ElastiCache (Redis OSS) version 5.0.4 (deprecated, use version 5.0.6)
- ElastiCache (Redis OSS) version 5.0.3 (deprecated, use version 5.0.6)
- ElastiCache (Redis OSS) version 5.0.0 (deprecated, use version 5.0.6)
- ElastiCache (Redis OSS) version 4.0.10 (enhanced)
- Past End of Life (EOL) versions (3.x)
- Past End of Life (EOL) versions (2.x)

ElastiCache (Redis OSS) version 7.1 (enhanced)

This release contains performance improvements which enable workloads to drive higher throughput and lower operation latencies. ElastiCache 7.1 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 (Redis OSS) version 7.0 (enhanced)

ElastiCache (Redis OSS) 7.0 adds a number of improvements and support for new functionality:

- Redis OSS 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: ElastiCache for Redis OSS 7 adds support for the next version of Redis OSS Access Control Lists (ACLs). With ElastiCache for Redis OSS 7, clients can now specify multiple sets of permissions on specific keys or keyspaces in Redis OSS.
- Sharded Pub/Sub: ElastiCache for Redis OSS 7 adds support to run Redis OSS Pub/Sub functionality in a sharded way when running ElastiCache in Cluster Mode Enabled (CME). Redis OSS Pub/Sub capabilities enable publishers to issue messages to any number of subscribers on a channel. With Amazon ElastiCache for Redis OSS 7, 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 Redis OSS version 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 for Redis OSS version 6.

For more information on the Redis OSS 7.0 release, see Redis OSS 7.0 Release Notes at Redis OSS on GitHub.

ElastiCache (Redis OSS) version 6.2 (enhanced)

ElastiCache (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.

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 Redis OSS.

You specify the engine version by using 6.2. ElastiCache (Redis OSS) will automatically invoke the preferred patch version of Redis OSS 6.2 that is available. For example, when you create/modify a cache 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 (Redis OSS) 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 (Redis OSS) version 6.0 (enhanced)

Amazon ElastiCache (Redis OSS) introduces the next version of 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 (Redis OSS) will offer a single version for each Redis OSS minor release, rather than offering multiple patch versions. ElastiCache (Redis OSS) will automatically manage the patch version of your running cache 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 (Redis OSS) 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 (Redis OSS) will automatically invoke the preferred patch version of Redis OSS 6.0 that is available. For example, when you create/modify a cache 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 (Redis OSS) version 5.0.6 (enhanced)

Amazon ElastiCache (Redis OSS) introduces the next version of 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 Introduction to Redis OSS Streams.
- Support for a family of stream commands, such as XADD, XRANGE and XREAD. For more information, see Redis OSS 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**
>
> Amazon ElastiCache for Redis OSS 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 (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 (Redis OSS) version 5.0.5 (deprecated, use version 5.0.6)

Amazon ElastiCache (Redis OSS) introduces the next version of the Redis OSS engine;. It includes online configuration changes for ElastiCache (Redis OSS) 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 (Redis OSS) version 5.0.4 (deprecated, use version 5.0.6)

Amazon ElastiCache (Redis OSS) introduces the next version of the Redis OSS engine supported by Amazon 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 (Redis OSS) version 5.0.3 (deprecated, use version 5.0.6)

Amazon ElastiCache (Redis OSS) introduces the next version of the Redis OSS engine supported by Amazon ElastiCache which includes bug fixes.

ElastiCache (Redis OSS) version 5.0.0 (deprecated, use version 5.0.6)

Amazon ElastiCache (Redis OSS) introduces the next major version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache (Redis OSS) 5.0.0 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 Introduction to Redis OSS Streams.
- Support for a family of stream commands, such as `XADD`, `XRANGE` and `XREAD`. For more information, see Redis OSS 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 (Redis OSS) version 4.0.10 (enhanced)

Amazon ElastiCache (Redis OSS) introduces the next major version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache (Redis OSS) 4.0.10 brings support the following improvements:

- Both online cluster resizing and encryption in a single ElastiCache (Redis OSS) version. For more information, see the following:
  - Scaling clusters in Redis OSS (Cluster Mode Enabled)
  - Online resharding and shard rebalancing for 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 Redis OSS 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 (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 Redis OSS version 3.2.10 can use the console to upgrade their clusters via online upgrade.

Comparing ElastiCache (Redis OSS) 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 (Redis OSS) version 3.2.10 (enhanced)

Amazon ElastiCache (Redis OSS) introduces the next major version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache (Redis OSS) 3.2.10 introduces online cluster resizing to add or remove shards from the cluster while it continues to serve incoming I/O requests. ElastiCache (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 (Redis OSS) versions 3.2.6 and 3.2.10

Feature	3.2.6	3.2.10
Online cluster resizing *	No	Yes
In-transit encryption **	Yes	No
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 and shard rebalancing for Redis OSS (cluster mode enabled)
- Online cluster resizing

ElastiCache (Redis OSS) version 3.2.6 (enhanced)

Amazon ElastiCache (Redis OSS) introduces the next major version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache (Redis OSS) 3.2.6 users have 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 (Redis OSS) version 3.2.4 (enhanced)

Amazon ElastiCache (Redis OSS) version 3.2.4 introduces the next major version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache (Redis OSS) 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
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 Redis OSS Commands: GEO on the Redis OSS 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 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 (Redis OSS) version 2.8.24 (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 (Redis OSS) version 2.8.23 (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 (Redis OSS) version 2.8.22 (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. For more information, see How synchronization and backup are implemented. 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 (Redis OSS) version 2.8.21

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 (Redis OSS) version 2.8.19

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 Redis OSS.

ElastiCache (Redis OSS) version 2.8.6

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 (Redis OSS) version 2.6.13

Redis OSS version 2.6.13 was the initial version of Redis OSS supported by Amazon ElastiCache (Redis OSS). Multi-AZ is not supported on Redis OSS 2.6.13.

Redis OSS versions 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 (Redis OSS) patch versions from 5.0.0 to 5.0.5 are deprecated. Use versions 5.0.6 or greater.

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 **Note** For US-ISO-EAST-1, US-ISO-WEST-1, and US-ISOB-EAST-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 **Note** For US-ISO-EAST-1, US-ISO-WEST-1, and US-ISOB-EAST-1 Regions, we recommend 5.0.6 or higher.	January 13, 2023

How to upgrade engine versions

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. For more information, see the following topics.

How to modify clusters and replication groups	
Clusters	**Replication groups**
Using the AWS Management Console	Using the AWS Management Console
Using the AWS CLI	Using the AWS CLI
Using the ElastiCache API	Using the ElastiCache API

Resolving blocked Redis OSS engine upgrades

As shown in the following table, your 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 Redis OSS engine upgrade

- Do one of the following:
  - Schedule your Redis OSS 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.

Major version behavior and compatibility differences

> **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 Redis OSS 6.0, ElastiCache (Redis OSS) will offer a single version for each Redis OSS minor release, rather than offering multiple patch versions. ElastiCache (Redis OSS) will automatically manage the patch version of your running cache 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 (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 (Redis OSS) versions are 7.0 and 6.2, ElastiCache (Redis OSS) will support versions 7.0 and 6.2 in the new AWS region. As newer MAJOR.MINOR versions of ElastiCache (Redis OSS) are released, ElastiCache will continue to add support for the newly released ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 5.0.
- 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.

- Inn Redis OSS 5.0.3 and above, ElastiCache (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 Redis OSS versions 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 (Redis OSS). We recommend using MemoryDB when data needs to be persisted durably.
- Starting in Redis OSS 2.8.22, ElastiCache (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 Redis OSS versions end of life schedule.

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

- Working with Redis OSS
- Best practices with Redis OSS clients
- Managing Reserved Memory
- Best practices when working with self-designed clusters
- Redis OSS best practices
- Caching strategies

Working with Redis OSS

Below you can find information about the Redis OSS interface within ElastiCache.

Topics

- Supported and restricted Redis OSS commands
- Redis OSS configuration and limits

Supported and restricted Redis OSS commands

Supported Redis OSS commands

Supported Redis OSS commands

The following Redis OSS commands are supported by serverless caches. In addition to these commands, these Supported Redis OSS JSON commands are also supported.

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.

  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 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

- `READONLY`

  Enables read-only queries for a connection to a Redis OSS Cluster replica node.

  Learn more

- `READWRITE`

  Enables read-write queries for a connection to a Redis OSS Cluster replica node.

  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 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
- `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

- `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

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.

  PUBSUB-SHARDCHANNELS
- `PUBSUB SHARDNUMSUB`

  Returns the count of subscribers of shard channels.

  PUBSUB-SHARDNUMSUB
- `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

- `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

- `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 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
- `SCARDT`

  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
- `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

Restricted 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, 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
- 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`
- `unwatch`
- `wait`
- `watch`

Redis OSS configuration and limits

The Redis OSS engine provides a number of configuration parameters, some of which are modifiable in ElastiCache (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 Redis OSS configuration is not modifiable. The following 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` `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.

Name	Details	Description
`client-query-buffer-limit`	1 GiB	The maximum size of a single client query buffer. Additionally, clients cannot issue a request with more than 4,000 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-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`.

Name	Details	Description
`lua-time-limit`	5000	The maximum execution time for a Lua script, in milliseconds, before ElastiCache takes action to stop the script. If `lua-time-limit` is exceeded, all Redis OSS commands may 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.
`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.
`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 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.

Name	Details	Description
`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
Key name length	4 KiB	The maximum size for a single 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 Redis OSS Lua script. Attempts to load a Lua script larger than this will receive an error.
Slot size	32 GiB	The maximum size of a single Redis OSS hash slot. Clients trying to set more data than this on a single Redis OSS slot will trigger the eviction policy on the slot, and if no keys are evictable, will receive an out of memory (`OOM`) error.

Self-designed clusters

For self-designed clusters, see [Redis OSS-specific 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.

Best practices with Redis OSS clients

Learn best practices for common scenarios and follow along with code examples of some of the most popular open source Redis OSS client libraries (redis-py, PHPRedis, and Lettuce).

Topics

Large number of connections

Serverless caches and individual ElastiCache (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. Redis OSS is 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 the 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 Redis OSS cluster with a Redis OSS 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 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 (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);
}
```

Redis OSS cluster client discovery and exponential backoff

When connecting to an ElastiCache (Redis OSS) cluster in cluster mode enabled, the corresponding Redis OSS 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 SLOT or CLUSTER NODE command to the Redis OSS server. We recommend the CLUSTER SLOT 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 SLOT 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 Redis OSS server unresponsive for a prolonged period of time, with the CPU utilization at 100%. The outage is prolonged if each CLUSTER SLOT 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 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 retries one time immediately after a failure. This mechanism can be enabled through the `retry_on_timeout` argument supplied when creating a Redis OSS object. Here we demonstrate a custom retry mechanism with exponential backoff and jitter. We've submitted a pull request to natively implement exponential backoff in redis-py (#1494). In the future it may not be necessary to implement manually.

```
def run_with_backoff(function, retries=5):
base_backoff = 0.1 # base 100ms backoff
max_backoff = 10 # sleep for maximum 10 seconds
tries = 0
while True:
try:
  return function()
except (ConnectionError, TimeoutError):
  if tries >= retries:
```

```
 raise
  backoff = min(max_backoff, base_backoff * (pow(2, tries) + random.random()))
  print(f"sleeping for {backoff:.2f}s")
  sleep(backoff)
  tries += 1
```

You can then use the following code to set a value:

```
client = redis.Redis(connection_pool=redis.BlockingConnectionPool(host=HOST,
 max_connections=10))
res = run_with_backoff(lambda: client.set("key", "value"))
print(res)
```

Depending on your workload, you might want to change the base backoff value from 1 second to a few tens or hundreds of milliseconds for latency-sensitive workloads.

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

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 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 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 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

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 Redis OSS-specific 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.

Redis OSS Lua scripts

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 Redis OSS.

Unparameterized Lua scripts

Each Lua script is cached on the Redis OSS server before it runs. Unparameterized Lua scripts are unique, which can lead to the 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.

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 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 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 Redis OSS even when Redis OSS is over `maxmemory`:

- The script starts when the 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 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

In some scenarios, an application may store large composite items in Redis OSS (such as a multi-GB hash dataset). This is not a recommended practice because it often leads to performance problems in 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 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

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

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 *dynamicRefreshResources* later in this topic.

SocketOption

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 Redis OSS 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 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 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 dynamicRefreshResources. We recommend enabling dynamicRefreshResources for small clusters, and disabling it for large clusters. dynamicRefreshResources 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

Configure DnsResolver with DirContextDnsResolver. The DNS resolver is based on Java's com.sun.jndi.dns.DnsContextFactory.

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()
   .dnsResolver(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();
```

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 configuration for cluster mode and 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
    .dnsResolver(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 optiontts
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 configuration for cluster mode disabled and 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()
        .dnsResolver(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);
```

IPv6 client examples

Following are best practices for interacting with IPv6 enabled ElastiCache resources with commonly used open-source client libraries. You can view existing best practices for interacting with

ElastiCache for recommendations on configuring clients for ElastiCache resources. However, there are some caveats worth noting when interacting with IPv6 enabled resources.

Validated clients

ElastiCache is compatible with open-source Redis OSS. This means that open source Redis OSS clients that support IPv6 connections should be able to connect to IPv6 enabled ElastiCache (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)

Validated Clients:

- Redis Py () – 4.1.2
- Lettuce – Version: 6.1.6.RELEASE
- Jedis – Version: 3.6.0

Configuring a preferred protocol for dual stack clusters

For cluster mode enabled 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 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 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;
}));
```

TLS enabled dual stack ElastiCache clusters

When TLS is enabled for ElastiCache clusters the cluster discovery functions (`cluster slots`, `cluster shards`, and `cluster nodes`) 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. **Including those to other 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 main 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

Reserved memory is memory set aside for nondata use. When performing a backup or failover, Redis OSS uses available memory to record write operations to your cluster while the cluster's data is being written to the .rdb file. If you don't have sufficient memory available for all the writes, the process fails. Following, you can find information on options for managing reserved memory for ElastiCache for Redis OSS and how to apply those options.

Topics

How Much Reserved Memory Do You Need?

If you are running a version of Redis OSS before 2.8.22, reserve more memory for backups and failovers than if you are running Redis OSS 2.8.22 or later. This requirement is due to the different ways that ElastiCache for Redis OSS implements the backup process. The rule of thumb is to reserve half of a node type's `maxmemory` value for Redis OSS overhead for versions before 2.8.22, and one-fourth for Redis OSS versions 2.8.22 and later.

Due to different ways that ElastiCache implements the backup and replication process, the rule of thumb is to reserve 25% of a node type's `maxmemory` value by using the `reserved-memory-percent` parameter. This is the default value and recommended for most cases.

When burstable micro and small instance types are operating near the `maxmemory` limits, they may experience swap usage. To improve the operational reliability on these instance types during backup, replication and high traffic, we recommend increasing the value of the `reserved-memory-percent` parameter up to 30% on small instance types, and up to 50% on micro instance types.

For write-heavy workloads on ElastiCache clusters with data tiering, we recommend increasing the `reserved-memory-percent` to up to 50% of the node's available memory.

For more information, see the following:

- Ensuring that you have enough memory to create a Redis OSS snapshot

- How synchronization and backup are implemented
- Data tiering

Parameters to Manage Reserved Memory

As of March 16, 2017, Amazon ElastiCache (Redis OSS) provides two mutually exclusive parameters for managing your Redis OSS memory, `reserved-memory` and `reserved-memory-percent`. Neither of these parameters is part of the Redis OSS distribution.

Depending upon when you became an ElastiCache customer, one or the other of these parameters is the default memory management parameter. This parameter applies when you create a new Redis OSS cluster or replication group and use a default parameter group.

- For customers who started before March 16, 2017 – When you create a Redis OSS cluster or replication group using the default parameter group, your memory management parameter is `reserved-memory`. In this case, zero (0) bytes of memory are reserved.
- For customers who started on or after March 16, 2017 – When you create a Redis OSS cluster or replication group using the default parameter group, your memory management parameter is `reserved-memory-percent`. In this case, 25 percent of your node's `maxmemory` value is reserved for nondata purposes.

After reading about the two Redis OSS memory management parameters, you might prefer to use the one that isn't your default or with nondefault values. If so, you can change to the other reserved memory management parameter.

To change the value of that parameter, you can create a custom parameter group and modify it to use your preferred memory management parameter and value. You can then use the custom parameter group whenever you create a new Redis OSS cluster or replication group. For existing clusters or replication groups, you can modify them to use your custom parameter group.

For more information, see the following:

- Specifying Your Reserved Memory Management Parameter
- Creating a parameter group
- Modifying a parameter group
- Modifying an ElastiCache cluster

- Modifying a replication group

The reserved-memory Parameter

Before March 16, 2017, all ElastiCache for Redis OSS reserved memory management was done using the parameter `reserved-memory`. The default value of `reserved-memory` is 0. This default reserves no memory for Redis OSS overhead and allows Redis OSS to consume all of a node's memory with data.

Changing `reserved-memory` so you have sufficient memory available for backups and failovers requires you to create a custom parameter group. In this custom parameter group, you set `reserved-memory` to a value appropriate for the Redis OSS version running on your cluster and cluster's node type. For more information, see How Much Reserved Memory Do You Need?

The ElastiCache for Redis OSS parameter `reserved-memory` is specific to ElastiCache for Redis OSS and isn't part of the Redis OSS distribution.

The following procedure shows how to use `reserved-memory` to manage the memory on your Redis OSS cluster.

To reserve memory using reserved-memory

1. Create a custom parameter group specifying the parameter group family matching the engine version you're running—for example, specifying the `redis2.8` parameter group family. For more information, see Creating a parameter group.

   ```
   aws elasticache create-cache-parameter-group \
      --cache-parameter-group-name redis6x-m3xl \
      --description "Redis OSS 2.8.x for m3.xlarge node type" \
      --cache-parameter-group-family redis6.x
   ```

2. Calculate how many bytes of memory to reserve for Redis OSS overhead. You can find the value of `maxmemory` for your node type at Redis OSS node-type specific parameters.

3. Modify the custom parameter group so that the parameter `reserved-memory` is the number of bytes you calculated in the previous step. The following AWS CLI example assumes you're running a version of Redis OSS before 2.8.22 and need to reserve half of the node's `maxmemory`. For more information, see Modifying a parameter group.

   ```
   aws elasticache modify-cache-parameter-group \
   ```

```
    --cache-parameter-group-name redis28-m3xl \
    --parameter-name-values "ParameterName=reserved-memory,
 ParameterValue=7130316800"
```

You need a separate custom parameter group for each node type that you use, because each node type has a different `maxmemory` value. Thus, each node type needs a different value for `reserved-memory`.

4. Modify your Redis OSS cluster or replication group to use your custom parameter group.

   The following CLI example modifies the cluster `my-redis-cluster` to use the custom parameter group `redis28-m3xl` beginning immediately. For more information, see [Modifying an ElastiCache cluster](#).

```
aws elasticache modify-cache-cluster \
    --cache-cluster-id my-redis-cluster \
    --cache-parameter-group-name redis28-m3xl \
    --apply-immediately
```

   The following CLI example modifies the replication group `my-redis-repl-grp` to use the custom parameter group `redis28-m3xl` beginning immediately. For more information, [Modifying a replication group](#).

```
aws elasticache modify-replication-group \
    --replication-group-id my-redis-repl-grp \
    --cache-parameter-group-name redis28-m3xl \
    --apply-immediately
```

The reserved-memory-percent parameter

On March 16, 2017, Amazon ElastiCache introduced the parameter `reserved-memory-percent` and made it available on all versions of ElastiCache (Redis OSS). The purpose of `reserved-memory-percent` is to simplify reserved memory management across all your clusters. It does so by enabling you to have a single parameter group for each parameter group family (such as `redis2.8`) to manage your clusters' reserved memory, regardless of node type. The default value for `reserved-memory-percent` is 25 (25 percent).

The ElastiCache for Redis OSS parameter `reserved-memory-percent` is specific to ElastiCache (Redis OSS) and isn't part of the Redis OSS distribution.

If your cluster is using a node type from the r6gd family and your memory usage reaches 75 percent, data-tiering will automatically be triggered. For more information, see Data tiering.

To reserve memory using reserved-memory-percent

To use `reserved-memory-percent` to manage the memory on your ElastiCache for Redis OSS cluster, do one of the following:

- If you are running Redis OSS 2.8.22 or later, assign the default parameter group to your cluster. The default 25 percent should be adequate. If not, take the steps described following to change the value.
- If you are running a version of Redis OSS before 2.8.22, you probably need to reserve more memory than `reserved-memory-percent`'s default 25 percent. To do so, use the following procedure.

To change the percent value of reserved-memory-percent

1. Create a custom parameter group specifying the parameter group family matching the engine version you're running—for example, specifying the `redis2.8` parameter group family. A custom parameter group is necessary because you can't modify a default parameter group. For more information, see Creating a parameter group.

   ```
   aws elasticache create-cache-parameter-group \
      --cache-parameter-group-name redis28-50 \
      --description "Redis OSS 2.8.x 50% reserved" \
      --cache-parameter-group-family redis2.8
   ```

   Because `reserved-memory-percent` reserves memory as a percent of a node's `maxmemory`, you don't need a custom parameter group for each node type.

2. Modify the custom parameter group so that `reserved-memory-percent` is 50 (50 percent). For more information, see Modifying a parameter group.

   ```
   aws elasticache modify-cache-parameter-group \
      --cache-parameter-group-name redis28-50 \
      --parameter-name-values "ParameterName=reserved-memory-percent,
    ParameterValue=50"
   ```

3. Use this custom parameter group for any Redis OSS clusters or replication groups running a version of Redis OSS older than 2.8.22.

The following CLI example modifies the Redis OSS cluster `my-redis-cluster` to use the custom parameter group `redis28-50` beginning immediately. For more information, see Modifying an ElastiCache cluster.

```
aws elasticache modify-cache-cluster \
   --cache-cluster-id my-redis-cluster \
   --cache-parameter-group-name redis28-50 \
   --apply-immediately
```

The following CLI example modifies the Redis OSS replication group `my-redis-repl-grp` to use the custom parameter group `redis28-50` beginning immediately. For more information, see Modifying a replication group.

```
aws elasticache modify-replication-group \
   --replication-group-id my-redis-repl-grp \
   --cache-parameter-group-name redis28-50 \
   --apply-immediately
```

Specifying Your Reserved Memory Management Parameter

If you were a current ElastiCache customer on March 16, 2017, your default reserved memory management parameter is `reserved-memory` with zero (0) bytes of reserved memory. If you became an ElastiCache customer after March 16, 2017, your default reserved memory management parameter is `reserved-memory-percent` with 25 percent of the node's memory reserved. This is true no matter when you created your ElastiCache (Redis OSS) cluster or replication group. However, you can change your reserved memory management parameter using either the AWS CLI or ElastiCache API.

The parameters `reserved-memory` and `reserved-memory-percent` are mutually exclusive. A parameter group always has one but never both. You can change which parameter a parameter group uses for reserved memory management by modifying the parameter group. The parameter group must be a custom parameter group, because you can't modify default parameter groups. For more information, see Creating a parameter group.

To specify reserved-memory-percent

To use `reserved-memory-percent` as your reserved memory management parameter, modify a custom parameter group using the `modify-cache-parameter-group` command. Use the `parameter-name-values` parameter to specify `reserved-memory-percent` and a value for it.

The following CLI example modifies the custom parameter group `redis32-cluster-on` so that it uses `reserved-memory-percent` to manage reserved memory. A value must be assigned to `ParameterValue` for the parameter group to use the `ParameterName` parameter for reserved memory management. For more information, see Modifying a parameter group.

```
aws elasticache modify-cache-parameter-group \
   --cache-parameter-group-name redis32-cluster-on \
   --parameter-name-values "ParameterName=reserved-memory-percent, ParameterValue=25"
```

To specify reserved-memory

To use `reserved-memory` as your reserved memory management parameter, modify a custom parameter group using the `modify-cache-parameter-group` command. Use the `parameter-name-values` parameter to specify `reserved-memory` and a value for it.

The following CLI example modifies the custom parameter group `redis32-m3xl` so that it uses `reserved-memory` to manage reserved memory. A value must be assigned to `ParameterValue` for the parameter group to use the `ParameterName` parameter for reserved memory management. Because the engine version is newer than 2.8.22, we set the value to `3565158400` which is 25 percent of a `cache.m3.xlarge`'s `maxmemory`. For more information, see Modifying a parameter group.

```
aws elasticache modify-cache-parameter-group \
   --cache-parameter-group-name redis32-m3xl \
   --parameter-name-values "ParameterName=reserved-memory, ParameterValue=3565158400"
```

Best practices when working with self-designed clusters

This section applies only when you choose to design your own Redis OSS clusters. We recommend you review and follow these best practices.

Topics

- Online cluster resizing
- Minimizing downtime during maintenance

Minimizing downtime with Multi-AZ

See Minimizing downtime in ElastiCache (Redis OSS) with Multi-AZ, to learn more about Multi-AZ and minimizing downtime.

Ensuring that you have enough memory to create a Redis OSS snapshot

Redis OSS snapshots and synchronizations in version 2.8.22 and later

Redis OSS 2.8.22 introduces 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. For more information, see How synchronization and backup are implemented.

Redis OSS snapshots and synchronizations before version 2.8.22

When you work with ElastiCache (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.

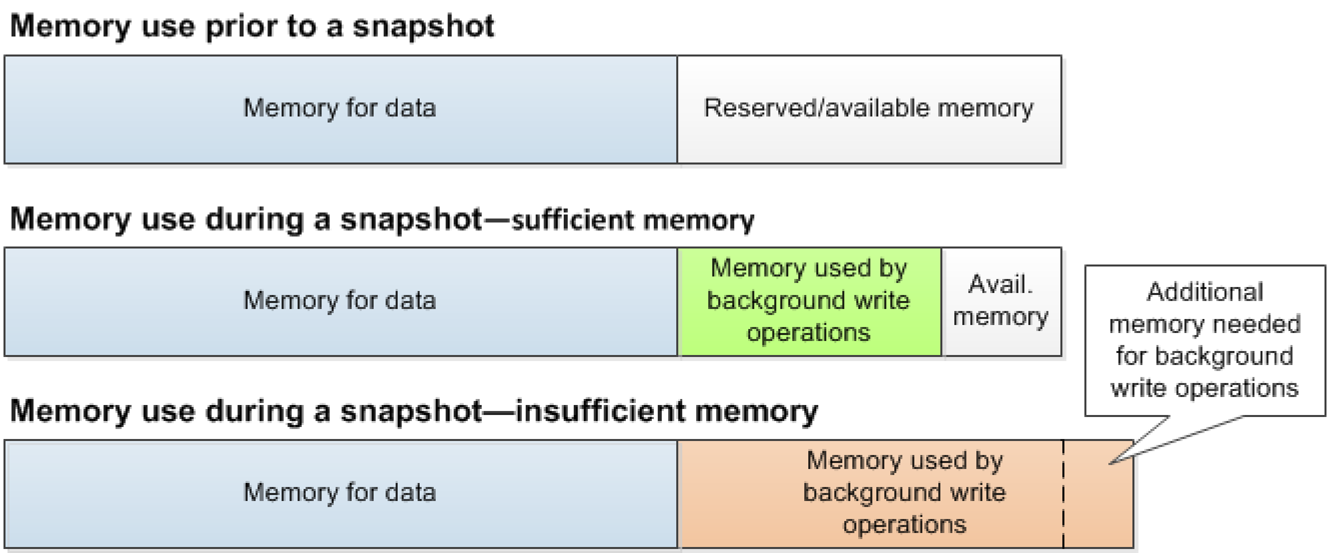
Background Write Process and Memory Usage

Whenever a background write process is called, Redis OSS forks its process (remember, Redis is 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 self-designed clusters](#).

For more information on how Redis OSS performs snapshots, see http://redis.io.

For more information on regions and Availability Zones, see [Choosing regions and availability zones](#).

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 Redis OSS record is updated and some new records are added to the cache.

Because of this, we recommend that you set `reserved-memory-percent` to 50 (50 percent) for Redis OSS versions before 2.8.22 or 25 (25 percent) for Redis OSS versions 2.8.22 and later.

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 Redis OSS uses on the box.

For more information on Redis-specific parameters in ElastiCache, see Redis OSS-specific parameters.

For information on creating and modifying parameter groups, see Creating a parameter group and Modifying a 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 (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 Redis OSS 7.0 or above and your application uses Redis OSS 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.
- Slots with large items are not migrated. In particular, slots with items larger than 256 MB post-serialization are not migrated.
- `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.

Redis OSS best practices

The following are best practices when using Redis OSS to improve performance and reliability:

- **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 (self-designed clusters), direct reads to replicas to achieve better scalability and/or lower latency. Reads from replicas are eventually consistent with the primary.

  In a self-designed 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 Redis OSS to support pub/sub workloads with high throughput, we recommend you use sharded pub/sub (available 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.

Caching strategies

In the following topic, you can find strategies for populating and maintaining your 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

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. Redis OSS can specify seconds or milliseconds for this value. 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 [Redis OSS `set` command](#) .

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 (Redis OSS)

Managing your self-designed cluster

This section contains topics that help you manage your self-designed clusters.

> **Note**
>
> These topics do not apply to ElastiCache Serverless.

Topics

Auto Scaling ElastiCache (Redis OSS) clusters

Prerequisites

ElastiCache (Redis OSS) Auto Scaling is limited to the following:

- Redis OSS (cluster mode enabled) clusters running Redis OSS engine version 6.0 onwards
- Data tiering (cluster mode enabled) clusters running Redis OSS engine version 7.0.7 onwards
- Instance sizes - Large, XLarge, 2XLarge
- Instance type families - R7g, R6g, R6gd, R5, M7g, M6g, M5, C7gn
- Auto Scaling in ElastiCache (Redis OSS) is not supported for clusters running in Global datastores, Outposts or Local Zones.

Managing Capacity Automatically with ElastiCache (Redis OSS) Auto Scaling

ElastiCache (Redis OSS) auto scaling is the ability to increase or decrease the desired shards or replicas in your ElastiCache (Redis OSS) service automatically. ElastiCache (Redis OSS) 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 (Redis OSS) auto scaling uses the policy to increase or decrease the number of instances in response to actual workloads.

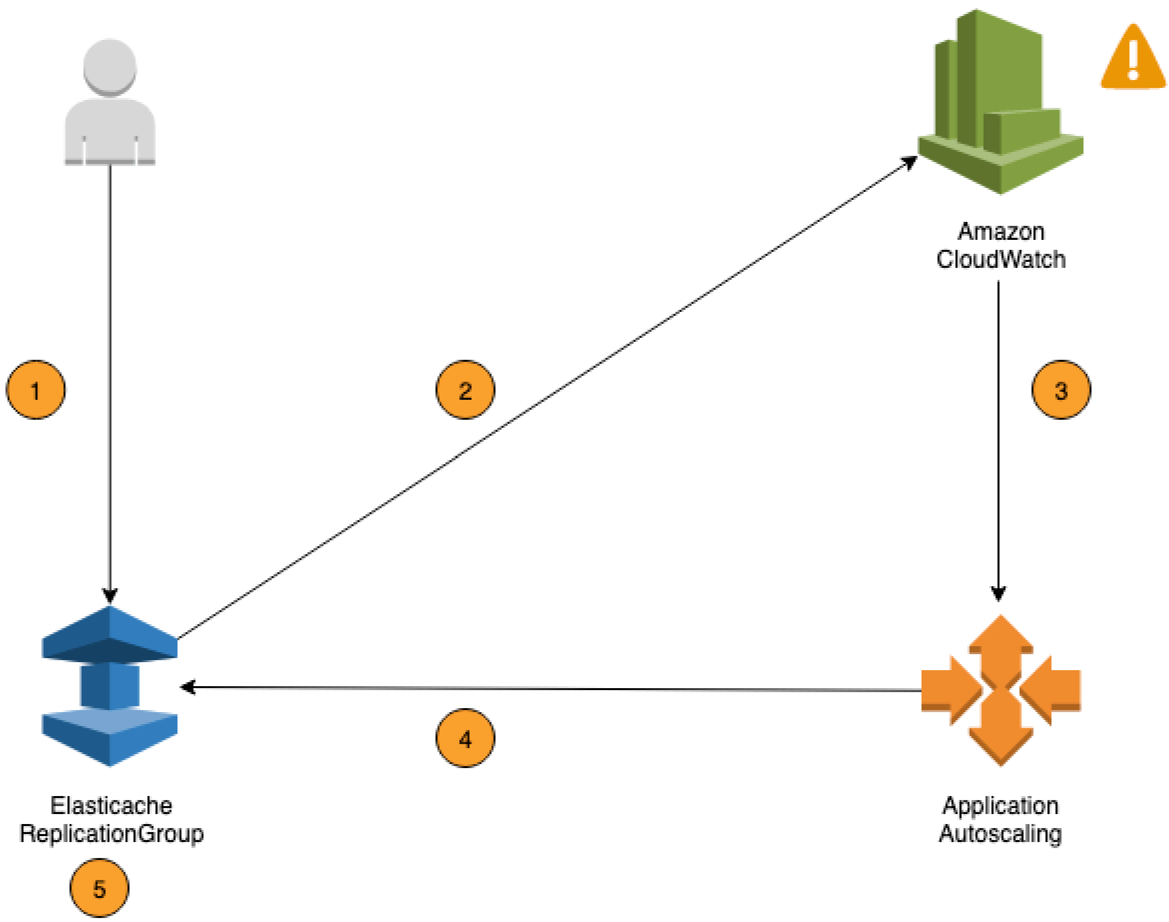
You can use the AWS Management Console to apply a scaling policy based on a predefined metric. A `predefined metric` is defined in an enumeration so that you can specify it by name in code or use it in the AWS Management Console. Custom metrics are not available for selection using the AWS Management 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 (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 (Redis OSS) auto scaling triggers scaling on your behalf.
- **Replicas** – Automatically add/remove replicas in the cluster similar to manual Increase/Decrease replica operations. ElastiCache (Redis OSS) auto scaling adds/removes replicas uniformly across all shards in the cluster.

ElastiCache (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 Application ElastiCache (Redis OSS) auto scaling – Increase or decrease the number of shards/replicas that your service runs based on the date and time.



The following steps summarize the ElastiCache (Redis OSS) auto scaling process as shown in the previous diagram:

1. You create an ElastiCache (Redis OSS) auto scaling policy for your ElastiCache (Redis OSS) Replication Group.
2. ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) auto scaling to evaluate your scaling policy.
4. ElastiCache (Redis OSS) auto scaling issues a Modify request to adjust your cluster capacity.

5. ElastiCache (Redis OSS) processes the Modify request, dynamically increasing (or decreasing) the cluster Shards/Replicas capacity so that it approaches your target utilization.

To understand how ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) Auto Scaling modifies resource settings only when the actual workload stays elevated (or depressed) for a sustained period of several minutes. The ElastiCache (Redis OSS) 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 (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 (Redis OSS) 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

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 (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 (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 ElastiCache (Redis OSS) 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 (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 ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 (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 ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) 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 (Redis OSS) shards/replicas as needed.

IAM Permissions Required for ElastiCache (Redis OSS) Auto Scaling

ElastiCache (Redis OSS) Auto Scaling is made possible by a combination of the ElastiCache (Redis OSS), CloudWatch, and Application Auto Scaling APIs. Clusters are created and updated with ElastiCache (Redis OSS), 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 (Redis OSS) Auto Scaling settings must have the appropriate permissions for the services that support dynamic scaling. IAM users must have permissions to use the actions shown in the following example policy:

```
{
    "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 (Redis OSS) auto scaling service also needs permission to describe your clusters and CloudWatch alarms, and permissions to modify your ElastiCache (Redis OSS) target capacity on your behalf. If you enable Auto Scaling for your ElastiCache (Redis OSS) cluster, it creates a service-linked role named `AWSServiceRoleForApplicationAutoScaling_ElastiCacheRG`. This service-linked role grants ElastiCache (Redis OSS) 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 (Redis OSS) auto scaling. For more information, see [Service-linked roles for ElastiCache (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 (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 family instance types), 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 Redis OSS 7.0 or above and your application uses Redis OSS 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

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

- Adding a scaling policy
- Registering a Scalable Target
- Defining a scaling policy
- Editing a scaling policy
- Deleting a scaling policy
- Use AWS 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 (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

Predefined Metric Name	CloudWatch Metric Name	CloudWatch Metric Dimension	Ineligible Instance Types
`ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage`	`DatabaseCapacityUsageCountedForEvictPercentage`	Redis OSS Replication Group Metrics	None
`ElastiCacheDatabaseMemoryUsageCountedForEvictPercentage`	`DatabaseMemoryUsageCountedForEvictPercentage`	Redis OSS Replication Group Metrics	R6gd

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 (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 (Redis OSS) 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 (Redis OSS) scales out by 30 percent, which results in 65 shards per cluster.

- if your Target breaches by 10 percent, ElastiCache (Redis OSS) 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 (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 (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 (Redis OSS) 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 (Redis OSS) cluster, provided that each of them uses a different metric. The intention of ElastiCache (Redis OSS) 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 (Redis OSS) Auto Scaling manages for a target tracking scaling policy. ElastiCache (Redis OSS) Auto Scaling deletes the alarms automatically when you delete the scaling policy.
- ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) Auto Scaling would not scale and be delayed til the capacity is available.
- ElastiCache (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 (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 **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 (Redis OSS) Auto Scaling policy is required to maintain.
12. For **Maximum capacity**, type the maximum number of shards that the ElastiCache (Redis OSS) 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 (Redis OSS) cluster, you register your cluster with ElastiCache (Redis OSS) auto scaling. You do so to define the scaling dimension and limits to be applied to that cluster. ElastiCache (Redis OSS) auto scaling dynamically scales the ElastiCache (Redis OSS) cluster along the `elasticache:replication-group:NodeGroups` scalable dimension, which represents the number of cluster shards.

Using the AWS CLI

To register your ElastiCache (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 ElastiCache (Redis OSS) cluster. For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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

By using predefined metrics, you can quickly define a target-tracking scaling policy for an ElastiCache (Redis OSS) cluster that works with target tracking in ElastiCache (Redis OSS) Auto Scaling.

Currently, ElastiCache (Redis OSS) supports the following predefined metrics in ElastiCache (Redis OSS) NodeGroup Auto Scaling:

- **ElastiCachePrimaryEngineCPUUtilization** – The average value of the `EngineCPUUtilization` metric in CloudWatch across all primary nodes in the ElastiCache (Redis OSS) cluster.
- **ElastiCacheDatabaseMemoryUsageCountedForEvictPercentage** – The average value of the `DatabaseMemoryUsageCountedForEvictPercentage` metric in CloudWatch across all primary nodes in the ElastiCache (Redis OSS) cluster.
- **ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage** – The average value of the `ElastiCacheDatabaseCapacityUsageCountedForEvictPercentage` metric in CloudWatch across all primary nodes in the ElastiCache (Redis OSS) 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 (Redis OSS) cluster. In this configuration, the `ElastiCachePrimaryEngineCPUUtilization` predefined metric is used to adjust the ElastiCache (Redis OSS) 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 (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": "RelicationGroup","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 (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 ElastiCache (Redis OSS) 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 (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 (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 (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 (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 ElastiCache (Redis OSS). For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) cluster.

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an ElastiCache (Redis OSS) cluster named `myscalablecluster` with ElastiCache (Redis OSS) 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 (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 the ElastiCache (Redis OSS). For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) cluster.

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an ElastiCache (Redis OSS) cluster named `myscalablecluster` with ElastiCache (Redis OSS) 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 (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 **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**.

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 (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 **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 (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 the ElastiCache (Redis OSS). For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the ElastiCache (Redis OSS) 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 an ElastiCache (Redis OSS) 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 (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 the ElastiCache (Redis OSS). For this parameter, the resource type is `ReplicationGroup` and the unique identifier is the name of the ElastiCache (Redis OSS) 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 an ElastiCache (Redis OSS) 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 AWS 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 (Redis OSS) to perform scaling activities at specific times. When you create a scheduled action, you specify an existing ElastiCache (Redis OSS) 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 (Redis OSS) 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 **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 **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 **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 AWS 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

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 (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 (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 (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 (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 (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 (Redis OSS) 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 (Redis OSS) cluster, provided that each of them uses a different metric. The intention of ElastiCache (Redis OSS) 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 (Redis OSS) Auto Scaling manages for a target tracking scaling policy. ElastiCache (Redis OSS) Auto Scaling deletes the alarms automatically when you delete the scaling policy or deleting the cluster.
- ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) Auto Scaling would not scale out until the capacity is available or if you manually modify the cluster to the instance types that have enough capacity.
- ElastiCache (Redis OSS) Auto Scaling doesn't support scaling of replicas with a cluster having `ReservedMemoryPercent` less than 25 percent. For more information, see Managing Reserved Memory.

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 an ElastiCache (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 **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 (Redis OSS) Auto Scaling policy is required to maintain.
12. For **Maximum capacity**, type the maximum number of replicas the ElastiCache (Redis OSS) 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 (Redis OSS) replication group with ElastiCache (Redis OSS) auto scaling.

Before you can use ElastiCache (Redis OSS) auto scaling with an ElastiCache (Redis OSS) cluster, you register your cluster with ElastiCache (Redis OSS) auto scaling. You do so to define the scaling dimension and limits to be applied to that cluster. ElastiCache (Redis OSS) auto scaling dynamically scales the ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 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 AWS 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 (Redis OSS) cluster

Using a predefined metric

By using predefined metrics, you can quickly define a target-tracking scaling policy for an ElastiCache (Redis OSS) cluster that works with target tracking in ElastiCache (Redis OSS) Auto Scaling. Currently, ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) cluster. The average value of the `EngineCPUUtilization` metric in CloudWatch across all replicas in the ElastiCache (Redis OSS) cluster. You can find the aggregated metric value in CloudWatch under ElastiCache (Redis OSS) `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 (Redis OSS) metric that changes in proportion to scaling. Not all ElastiCache (Redis OSS) 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 an ElastiCache (Redis OSS) 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": "RelicationGroup","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 an ElastiCache (Redis OSS) 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 (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 an ElastiCache (Redis OSS) 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 (Redis OSS) cluster

After registering your cluster with ElastiCache (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 (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 (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 ElastiCache (Redis OSS) cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) cluster.

Example

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an ElastiCache (Redis OSS) cluster named `myscalablecluster` with ElastiCache (Redis OSS) 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 (Redis OSS) 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 ElastiCache (Redis OSS) cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the ElastiCache (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 ElastiCache (Redis OSS) cluster.

Example

In the following example, you apply a target-tracking scaling policy named `scalablepolicy` to an ElastiCache (Redis OSS) cluster named `myscalablecluster` with ElastiCache (Redis OSS) 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 **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 (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 **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 (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 ElastiCache (Redis OSS) cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the ElastiCache 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 (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 ElastiCache (Redis OSS) cluster. For this parameter, the resource type is ReplicationGroup and the unique identifier is the name of the ElastiCache 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 an ElastiCache (Redis OSS) 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 AWS 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 (Redis OSS) to perform scaling activities at specific times. When you create a scheduled action, you specify an existing ElastiCache (Redis OSS) 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 (Redis OSS) 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 AWS 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

Redis OSS is a distributed in-memory database that supports sharding and replication. ElastiCache (Redis OSS) clusters are the distributed implementation of Redis OSS that allows data to be partitioned across multiple Redis OSS nodes. An ElastiCache (Redis OSS) cluster has two modes of operation, Cluster mode enabled (CME) and cluster mode disabled (CMD). In CME, Redis OSS works as a distributed database with multiple shards and nodes, while in CMD, Redis OSS works 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 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 Redis OSS engine version required for migration is 7.0.

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 (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 parameter groups

1. **Prepare** – Create a test CME cluster and make sure your stack is ready to work with it. ElastiCache (Redis OSS) has no way to verify your readiness. For more information, see Creating a cluster.
2. **Modify existing CMD Cluster Configuration to cluster mode compatible** – In this mode, there will be a single shard deployed, and ElastiCache (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 Redis OSS Cluster protocol and configuration endpoint. To change the 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 Redis OSS cluster mode to cluster mode compatible, the DescribeReplicationGroups API will return the ElastiCache (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.
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 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 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 self-designed clusters only.

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 (Redis OSS) 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

Each *global datastore* is a collection of one or more clusters that replicate to one another.

A global datastore consists of the following:

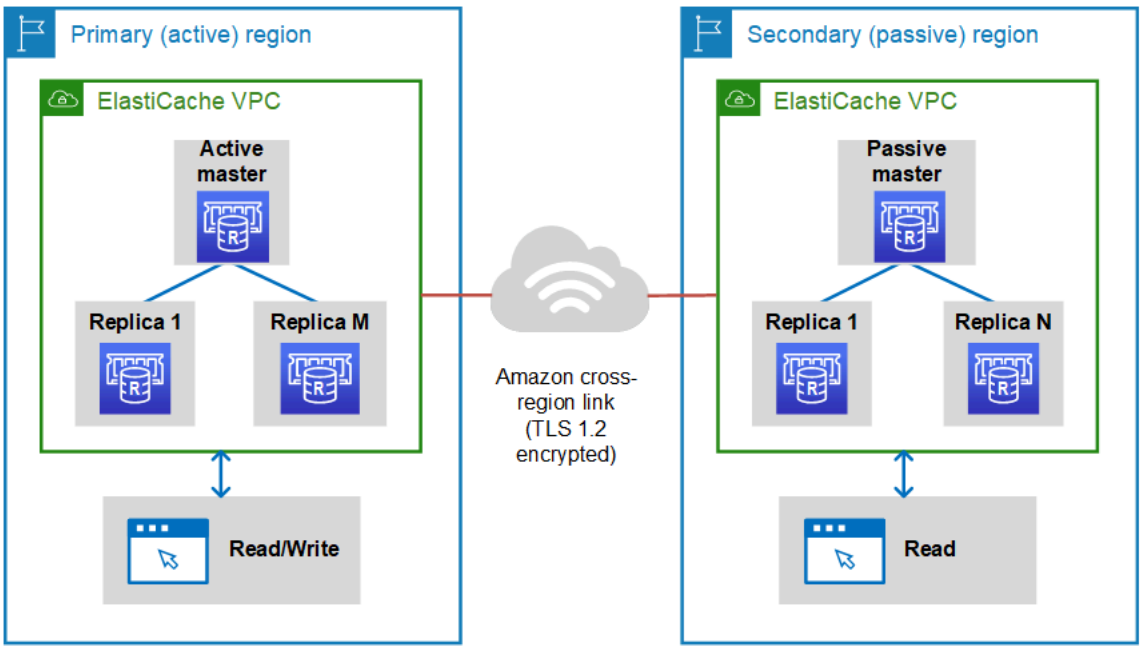
- **Primary (active) cluster** – A primary cluster accepts writes that are replicated to all clusters within the global datastore. A primary cluster also accepts read requests.
- **Secondary (passive) cluster** – A secondary cluster only accepts read requests and replicates data updates from a primary cluster. A secondary cluster needs to be in a different AWS Region than the primary cluster.

When you create a global datastore in ElastiCache, ElastiCache (Redis OSS) automatically replicates your data from the primary cluster to the secondary cluster. You choose the AWS Region where the 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 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: Asia Pacific (Seoul, Tokyo, Singapore, Sydney, Mumbai, and Osaka), Europe (Frankfurt, Paris, London, Ireland, and Stockholm), US East (N. Virginia and Ohio), US West (N. California and Oregon), South America (São Paulo), AWS GovCloud (US-West and US-East), Canada (Central) Region, China (Beijing and Ningxia)
- 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.

- Global datastores are not supported on instances older than m5 or r5.
- 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.

- You can work with global datastores only in VPC clusters. For more information, see Access Patterns for Accessing an ElastiCache Cache in an Amazon VPC. Global datastores aren't supported when you use EC2-Classic. For more information, see EC2-Classic in the *Amazon EC2 User Guide.*

> **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 (Redis OSS) .
- Global datastores support encryption at rest, encryption in transit, and Redis OSS 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.*

> **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 Redis OSS 5.0.6 or later.
2. Add up to two secondary clusters in different AWS Regions, again using the Redis OSS 5.0.6 engine or later.

The following procedures guide you on how to create a global datastore for Redis OSS and perform other operations using the ElastiCache (Redis OSS) console.

Topics

- [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 the Redis OSS 5.0.6 engine 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.
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 Redis OSS engine version 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.
  - **Redis OSS AUTH Default User** – An authentication mechanism for Redis OSS server. For more information, see Redis OSS 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)
   - Redis OSS 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 Redis OSS 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 Redis OSS-specific parameters and Creating a 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 an r6gd node type, data-tiering is automatically enabled. For more information, see Data tiering.

e. If you are creating a 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 Redis OSS (cluster mode enabled) cluster:

i. For **Number of shards**, choose the number of shards (partitions/node groups) that you want for this Redis OSS (cluster mode enabled) cluster.

   For some versions of Redis OSS (cluster mode enabled), you can change the number of shards in your cluster dynamically:

   - **Redis OSS 3.2.10 and later –** If your cluster is running Redis OSS 3.2.10 or later versions, you can change the number of shards in your cluster dynamically. For more information, see Scaling clusters in Redis OSS (Cluster Mode Enabled).
   - **Other Redis OSS versions –** If your cluster is running a version of Redis OSS before version 3.2.10, there's another approach. To change the number of shards in your cluster in this case, create a new cluster with the new number of shards. For more information, see Restoring from a backup into a new cache.

ii. For **Replicas per shard**, choose the number of read replica nodes that you want in each shard.

   The following restrictions exist for 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 Seeding a new self-designed 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.

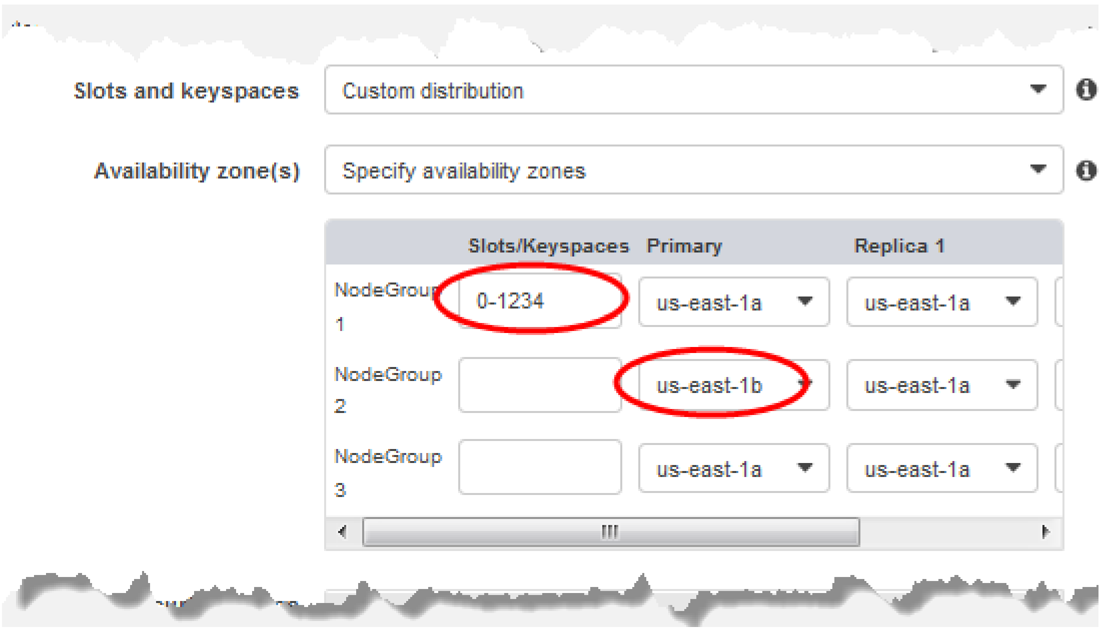
   For more information see, Create a subnet in your VPC.

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.



Specifying Keyspaces and Availability Zones

8. Choose **Next**

9. 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 Redis OSS engine version 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.
  - **Redis OSS AUTH Default User –** An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.
- **Redis OSS AUTH –** An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.

> **Note**
>
> For Redis OSS versions between 3.2.6 onward, excluding version 3.2.10, Redis OSS 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 Redis OSS 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 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 Redis OSS engine version 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.
  - **Redis OSS AUTH Default User –** An authentication mechanism for Redis OSS server. For more information, see Redis OSS AUTH.

> **Note**
>
> For Redis OSS versions between 4.0.2, when Encryption in-transit was first supported, and 6.0.4, Redis OSS 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)
- Redis OSS 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 Redis OSS 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
- **Redis OSS Engine Version:** The 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
- **Redis OSS 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 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
- Redis OSS 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)
    - Redis OSS 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 Redis OSS engine version 5.0.6 or later is 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 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 Redis OSS console or the ModifyCacheCluster API operation. For more information, see Modifying a 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 (Redis OSS) 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 `us-east-2`	`fpkhr`
US East (N. Virginia) Region `us-east-1`	`ldgnf`
US West (N. California) Region `us-west-1`	`virxk`
US West (Oregon) Region `us-west-2`	`sgaui`
Canada (Central) Region `ca-central-1`	`bxodz`
Asia Pacific (Mumbai) Region `ap-south-1`	`erpgt`
Asia Pacific (Tokyo) Region `ap-northeast-1`	`quwsw`

Region Name/Region	Prefix
Asia Pacific (Seoul) Region `ap-northeast-2`	`lfqnh`
Asia Pacific (Osaka) Region `ap-northeast-3`	`nlapn`
Asia Pacific (Singapore) Region `ap-southeast-1`	`vlqxn`
Asia Pacific (Sydney) Region `ap-southeast-2`	`vbgxd`
Europe (Frankfurt) Region `eu-central-1`	`iudkw`
Europe (Ireland) Region `eu-west-1`	`gxeiz`
Europe (London) Region `eu-west-2`	`okuqm`
EU (Paris) Region `eu-west-3`	`fgjhi`
South America (São Paulo) Region `sa-east-1`	`juxlw`
China (Beijing) Region `cn-north-1`	`emvgo`

Region Name/Region	Prefix
China (Ningxia) Region `cn-northwest-1`	`ckbem`
Asia Pacific (Hong Kong) Region `ap-east-1`	`knjmp`
AWS GovCloud (US-West) `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 (Redis OSS) 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
```

- 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 yes \
   --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 yes \
   --global-replication-group-id global-replication-group-name \
   --node-group-count 3
```

- rebalance-shards-in-global-replication-group

```
aws elasticache rebalance-shards-in-global-replication-group \
   --apply-immediately yes \
   --global-replication-group-id global-replication-group-name
```

Use help to list all available commands ElastiCache (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 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.

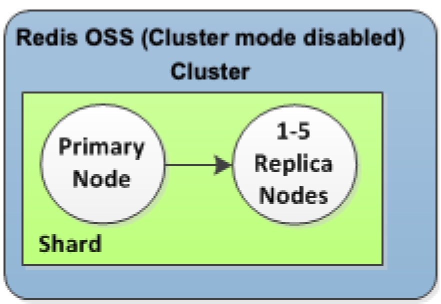
However, if you're running the Redis OSS engine, you can group 2 to 6 nodes into a cluster with replicas where 1 to 5 read-only nodes contain replicate data of the group's single read/write primary node. In this scenario, if one node fails for any reason, you do not lose all your data since it is replicated 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 Redis OSS cluster. Redis OSS (cluster mode disabled) clusters always have one shard. 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 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**.



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 Redis OSS](#).

Topics

Understanding 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—Redis OSS (cluster mode disabled)
- With data partitioned across up to 500 shards—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.

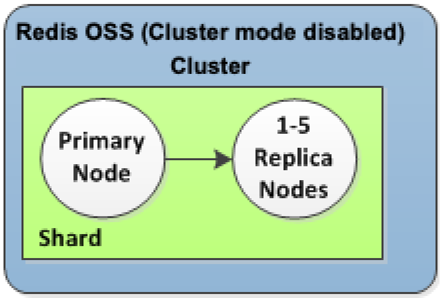
To request a limit increase, see AWS Service Limits and choose the limit type **Nodes per cluster per instance type**.

Topics

- Redis OSS (Cluster Mode Disabled)
- Redis OSS (cluster mode enabled)

Redis OSS (Cluster Mode Disabled)

A Redis OSS (cluster mode disabled) cluster has a single shard, inside of which is a collection of Redis OSS 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.



Redis OSS (cluster mode disabled) cluster with a single shard and replica nodes

You can use Redis OSS (cluster mode disabled) clusters with replica nodes to scale your Redis OSS 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 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 Redis OSS (cluster mode disabled) cluster with replicas. For more information, see Minimizing downtime in ElastiCache (Redis OSS) with Multi-AZ.

You can change the roles of the nodes within the 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 Redis OSS (cluster mode disabled) replication groups.

Redis OSS (cluster mode enabled)

A 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.

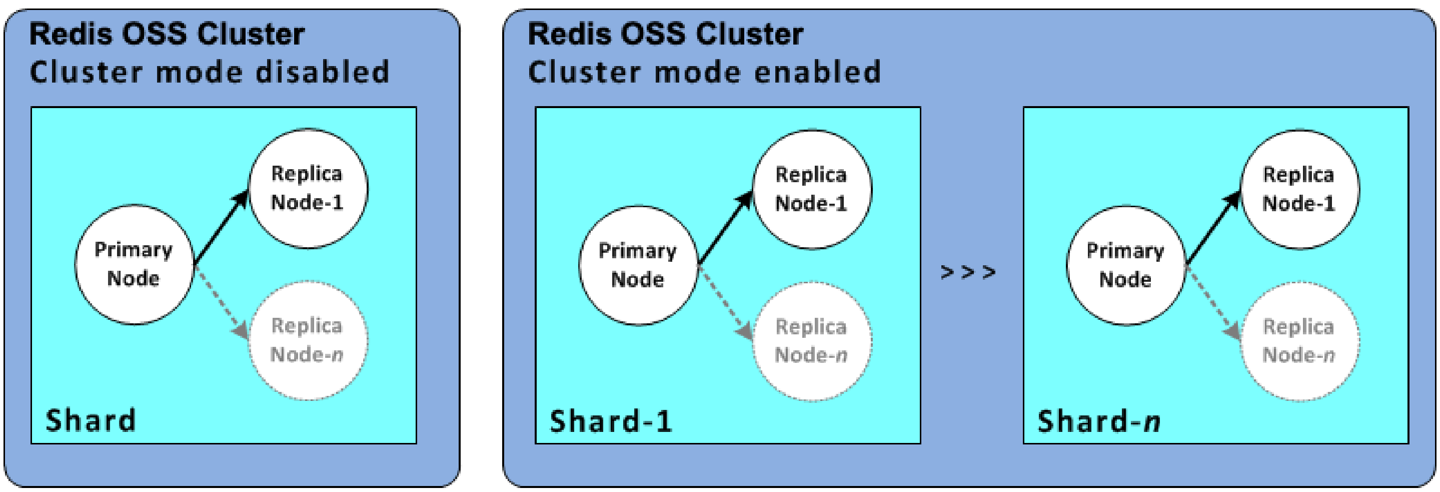
The node or shard limit can be increased to a maximum of 500 per cluster if the Redis OSS engine version is 5.0.6 or higher. For example, you can choose to configure a 500 node cluster that ranges between 83 shards (one primary and 5 replicas per shard) and 500 shards (single primary and no replicas). 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**.

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 Redis OSS (cluster mode enabled) cluster.

Applications use the Redis OSS (cluster mode enabled) cluster's *configuration endpoint* to connect with the nodes in the cluster. For more information, see Finding connection endpoints.



Redis OSS (cluster mode enabled) cluster with multiple shards and replica nodes

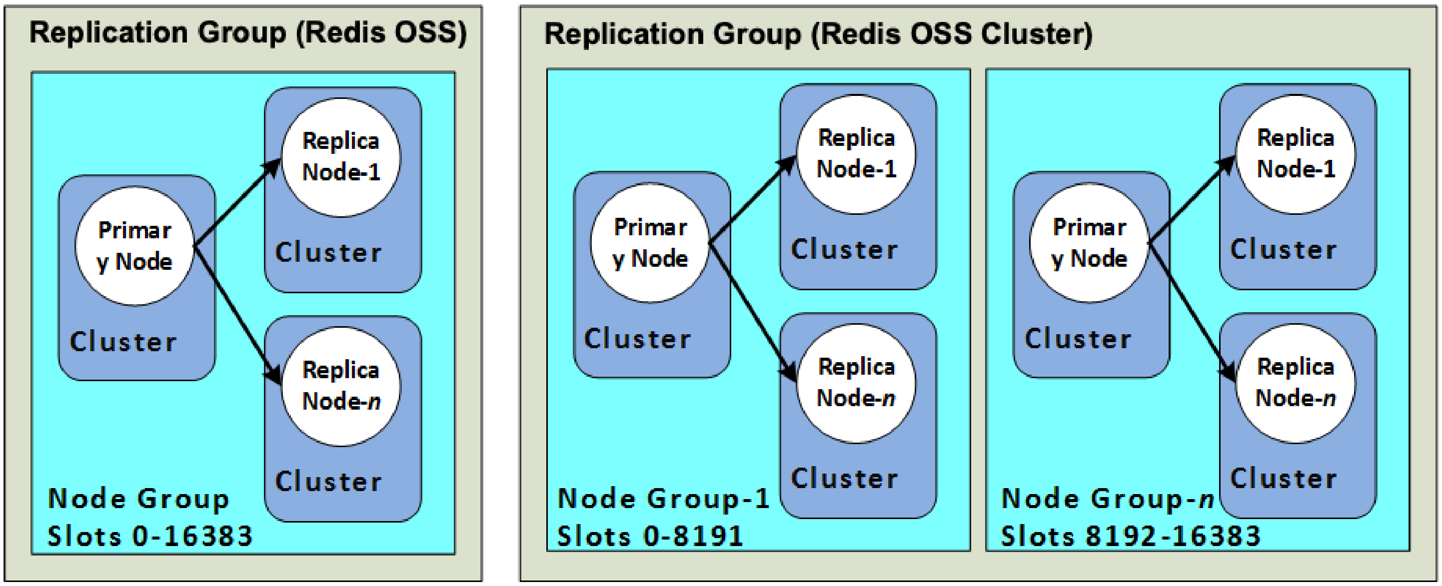
All of the nodes in a Redis OSS (cluster mode enabled) cluster must reside in the same region. To improve fault tolerance, you can provision both primaries and read replicas in multiple Availability Zones within that region.

Currently, in Redis OSS (cluster mode enabled), there are some limitations.

- You cannot manually promote any of the replica nodes to primary.

Replication: Redis OSS (Cluster Mode Disabled) vs. Redis OSS (Cluster Mode Enabled)

Beginning with Redis OSS version 3.2, you have the ability to create one of two distinct types of Redis OSS clusters (API/CLI: replication groups). A Redis OSS (cluster mode disabled) cluster always has a single shard (API/CLI: node group) with up to 5 read replica nodes. A Redis OSS (cluster mode enabled) cluster has up to 500 shards with 1 to 5 read replica nodes in each.



Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled) clusters

The following table summarizes important differences between Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled) clusters.

Comparing Redis OSS (Cluster Mode Disabled) and Redis OSS (Cluster Mode Enabled) Clusters

Feature	Redis OSS (cluster mode disabled)	Redis OSS (cluster mode enabled)
Modifiable	Yes. Supports adding and deleting replica nodes, and scaling up node type.	Limited. For more information, see Engine versions and upgrading and Scaling clusters in Redis OSS (Cluster Mode Enabled).
Data Partitioning	No	Yes

Feature	Redis OSS (cluster mode disabled)	Redis OSS (cluster mode enabled)
Shards	1	1 to 500
Read replicas	0 to 5 **Important** If you have no replicas and the node fails, you experience total data loss.	0 to 5 per shard. **Important** 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. Optional. On by default.	Yes 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 Redis OSS (cluster mode disabled) cluster.	Yes, using .rdb files from either a Redis OSS (cluster mode disabled) or a Redis OSS (cluster mode enabled) cluster.
Supported by	All Redis OSS versions	Redis OSS 3.2 and following
Engine upgradeable	Yes, with some limits. For more information, see Engine versions and upgrading .	Yes, with some limits. For more information, see Engine versions and upgrading .
Encryption	Versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule) and 4.0.10 and later.	Versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule) and 4.0.10 and later.

Feature	Redis OSS (cluster mode disabled)	Redis OSS (cluster mode enabled)
HIPAA Eligible	Versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule) and 4.0.10 and later.	Versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule) and 4.0.10 and later.
PCI DSS Compliant	Versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule) and 4.0.10 and later.	Versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule) and 4.0.10 and later.
Online resharding	N/A	Version 3.2.10 (scheduled for EOL, see Redis OSS versions end of life schedule) and later.

Which should I choose?

When choosing between Redis OSS (cluster mode disabled) 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. 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 Redis OSS (Cluster Mode Disabled) clusters with replica nodes.

  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 clusters in Redis OSS (Cluster Mode Enabled).

- **Node size v. number of nodes –** Because a 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 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 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 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 (Redis OSS) with Multi-AZ

There are a number of instances where ElastiCache (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 ElastiCache (Redis OSS) Cluster, the planned node replacements complete while the cluster serves incoming write requests.
- For 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 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.

Enabling ElastiCache Multi-AZ on your 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 Redis OSS clusters that have more than one node in each shard.

Topics

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 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 Redis OSS replication group. For information about adding a read replica to a cluster with replication, see Adding a read replica, for Redis OSS (Cluster Mode Disabled) replication groups.

Topics

Enabling Multi-AZ (Console)

You can enable Multi-AZ using the ElastiCache console when you create a new Redis OSS cluster or by modifying an existing Redis OSS cluster with replication.

Multi-AZ is enabled by default on 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 Redis OSS (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 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

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 Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding the Endpoints for Replication Groups (AWS CLI)
- Finding Endpoints for 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 Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding the Endpoints for Replication Groups (AWS CLI)
- Finding Endpoints for 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 Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding the Endpoints for Replication Groups (AWS CLI)
- Finding Endpoints for 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 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.

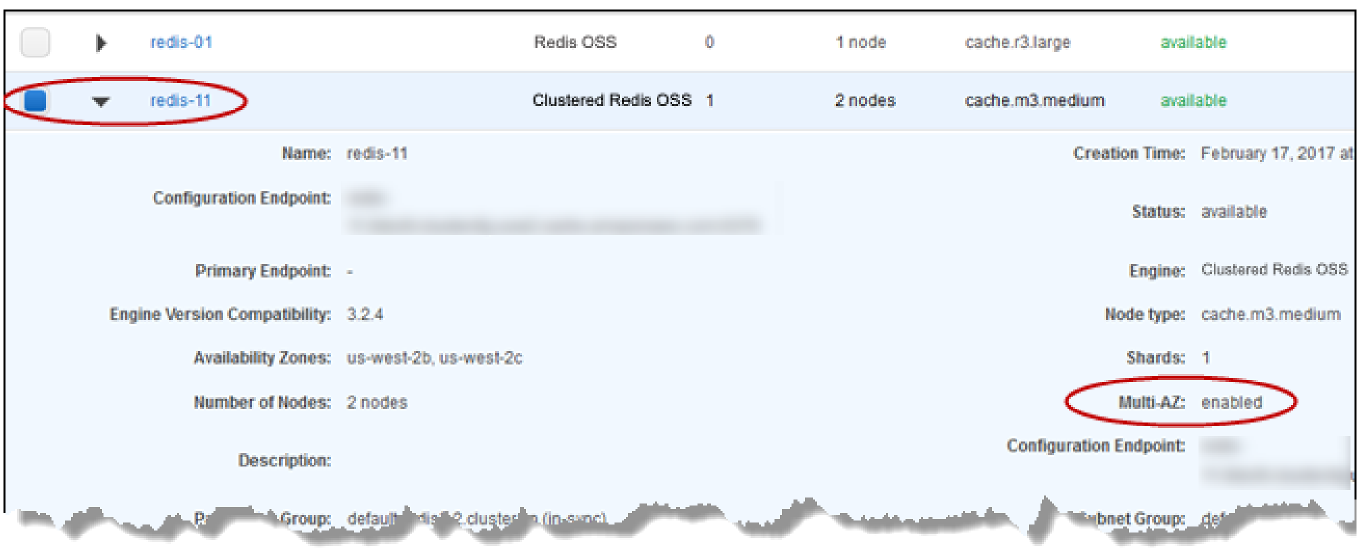
Topics

Testing automatic failover using the AWS Management Console

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 **Redis OSS**.
3. From the list of Redis OSS 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 AWS Management Console.



5. For Redis OSS (cluster mode disabled), choose the cluster's name.

   For 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 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 redis00-0003
```

For Windows:

```
aws elasticache test-failover ^
```

```
    --replication-group-id redis00 ^
    --node-group-id redis00-0003
```

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.m3.medium",
        "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=redis00-0003
    &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 Redis OSS Multi-AZ

Be aware of the following limitations for Redis OSS Multi-AZ:

- Multi-AZ is supported on Redis OSS version 2.8.6 and later.
- Redis OSS Multi-AZ isn't supported on T1 node types.
- 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.

When choosing the replica to promote to primary, ElastiCache (Redis OSS) 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 Redis OSS (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:
  1. Disable Multi-AZ on the cluster.
  2. Disable automatic failover on the cluster. You can do this using the Redis OSS console by clearing the **Auto failover** check box for the replication group. You can 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 (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 Redis OSS and isn't unique to ElastiCache Multi-AZ. For details about this Redis OSS behavior, see Replication on the Redis OSS website.

Important

For Redis OSS version 2.8.22 and later, you can't create external replicas.

For Redis OSS versions before 2.8.22, we recommend that you don't connect an external Redis OSS replica to an ElastiCache (Redis OSS) cluster that is Multi-AZ enabled. This unsupported configuration can create issues that prevent ElastiCache from properly performing failover and recovery. To connect an external Redis OSS replica to an ElastiCache cluster, make sure that Multi-AZ isn't enabled before you make the connection.

How synchronization and backup are implemented

All supported versions of Redis OSS support backup and synchronization between the primary and replica nodes. However, the way that backup and synchronization is implemented varies depending on the Redis OSS version.

Redis OSS Version 2.8.22 and Later

Redis OSS replication, in versions 2.8.22 and later, choose between two methods. For more information, see Redis OSS Versions Before 2.8.22 and Snapshot and restore.

During the forkless process, if the write loads are heavy, writes to the cluster are delayed to ensure that you don't accumulate too many changes and thus prevent a successful snapshot.

Redis OSS Versions Before 2.8.22

Redis OSS backup and synchronization in versions before 2.8.22 is a three-step process.

1. Fork, and in the background process, serialize the cluster data to disk. This creates a point-in-time snapshot.
2. In the foreground, accumulate a change log in the *client output buffer*.

   > **Important**
   >
   > If the change log exceeds the *client output buffer* size, the backup or synchronization fails. For more information, see Ensuring that you have enough memory to create a Redis OSS snapshot.

3. Finally, transmit the cache data and then the change log to the replica node.

Creating a Redis OSS replication group

You have the following options for creating a cluster with replica nodes. One applies when you already have an available 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 Redis OSS (cluster mode enabled) cluster must be created from scratch.

Option 1: Creating a Replication Group Using an Available Redis OSS (Cluster Mode Disabled) Cluster

Use this option to leverage an existing single-node 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 Available Redis OSS (Cluster Mode Disabled) Cluster.

> **Important**
>
> You cannot create a Redis OSS (cluster mode enabled) cluster using an existing cluster. To create a Redis OSS (cluster mode enabled) cluster (API/CLI: replication group) using the ElastiCache console, see Creating a Redis OSS (cluster mode enabled) cluster (Console).

Option 2: Creating a Redis OSS replication group from scratch

Use this option if you don't already have an available Redis OSS (cluster mode disabled) cluster to use as the cluster's primary node, or if you want to create a Redis OSS (cluster mode enabled) cluster. See Creating a Redis OSS replication group from scratch.

Creating a Replication Group Using an Available Redis OSS (Cluster Mode Disabled) Cluster

An available cluster is an existing single-node Redis OSS cluster. Currently, 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 Redis OSS (cluster mode enabled) cluster, see Creating a Redis OSS (Cluster Mode Enabled) cluster (Console).

The following procedure can only be used if you have a Redis OSS (cluster mode disabled) single-node cluster. This cluster's node becomes the primary node in the new cluster. If you do not have a Redis OSS (cluster mode disabled) cluster that you can use as the new cluster's primary, see Creating a Redis OSS replication group from scratch.

Creating a Replication Group Using an Available Redis OSS Cluster (Console)

See the topic Using the AWS Management Console.

Creating a replication group using an available Redis OSS cache cluster (AWS CLI)

There are two steps to creating a replication group with read replicas when using an available 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`.

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 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 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 cache 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 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`.

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 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 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 cache 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 Redis OSS replication group from scratch

Following, you can find how to create a Redis OSS replication group without using an existing Redis OSS cluster as the primary. You can create a Redis OSS (cluster mode disabled) 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 Redis OSS (cluster mode disabled) or a Redis OSS (cluster mode enabled) replication group. For guidance in deciding, see Replication: Redis OSS (Cluster Mode Disabled) vs. Redis OSS (Cluster Mode Enabled).

Topics

Creating a Redis OSS (Cluster Mode Disabled) replication group from scratch

You can create a Redis OSS (cluster mode disabled) replication group from scratch using the ElastiCache console, the AWS CLI, or the ElastiCache API. A Redis OSS (cluster mode disabled) replication group always has one node group, a primary cluster, and up to five read replicas. 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 Redis OSS (cluster mode disabled) replication group from scratch, take one of the following approaches:

Creating a Redis OSS (Cluster Mode Disabled) replication group from scratch (AWS CLI)

The following procedure creates a Redis OSS (cluster mode disabled) replication group using the AWS CLI.

When you create a 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.

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 current generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For more information on performance details for each node type, see [Amazon EC2 Instance Types](#).

--data-tiering-enabled

Set this parameter if you are using an r6gd node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see [Data tiering](#).

--cache-parameter-group

Specify a parameter group that corresponds to your engine version. If you are running Redis OSS 3.2.4 or later, specify the `default.redis3.2` parameter group or a parameter group derived from `default.redis3.2` to create a Redis OSS (cluster mode disabled) replication group. For more information, see [Redis OSS-specific 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 Redis OSS version 3.2.6 or 4.0.10.
- 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 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 Redis OSS (cluster mode disabled) replication group from scratch (ElastiCache API)

The following procedure creates a Redis OSS (cluster mode disabled) replication group using the ElastiCache API.

When you create a 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.

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 current generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For more information on performance details for each node type, see [Amazon EC2 Instance Types](#).

--data-tiering-enabled

Set this parameter if you are using an r6gd node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see [Data tiering](#).

CacheParameterGroup

Specify a parameter group that corresponds to your engine version. If you are running Redis OSS 3.2.4 or later, specify the `default.redis3.2` parameter group or a parameter group derived from `default.redis3.2` to create a Redis OSS (cluster mode disabled) replication group. For more information, see [Redis OSS-specific 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 Redis OSS version 3.2.6 or 4.0.10.
- 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 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 Redis OSS (Cluster Mode Enabled) from scratch

You can create a Redis OSS (cluster mode enabled) cluster (API/CLI: *replication group*) using the ElastiCache console, the AWS CLI, or the ElastiCache API. A 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 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 Redis OSS (Cluster Mode Enabled)

- Creating a Redis OSS (Cluster Mode Enabled) cluster (Console)
- Creating a Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)
- Creating a replication group in Redis OSS (Cluster Mode Enabled) from scratch (ElastiCache API)

Creating a Redis OSS (Cluster Mode Enabled) cluster (Console)

To create a Redis OSS (cluster mode enabled) cluster, see Creating a 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 Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)

The following procedure creates a Redis OSS (cluster mode enabled) replication group using the AWS CLI.

When you create a 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.

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 current generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For more information on performance details for each node type, see [Amazon EC2 Instance Types](#).

--data-tiering-enabled

Set this parameter if you are using an r6gd node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see [Data tiering](#).

--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 Redis OSS (cluster mode enabled) replication group. For more information, see [Redis OSS 6.x parameter changes](#).

--engine

redis

--engine-version

3.2.4

--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 Redis OSS version 3.2.6 or 4.0.10.
- 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 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 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 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 Redis OSS (Cluster Mode Enabled) from scratch (ElastiCache API)

The following procedure creates a Redis OSS (cluster mode enabled) replication group using the ElastiCache API.

When you create a 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.

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 current generation types provide more memory and computational power at lower cost when compared to their equivalent previous generation counterparts.

For more information on performance details for each node type, see [Amazon EC2 Instance Types](https://aws.amazon.com/ec2/instance-types/).

--data-tiering-enabled

Set this parameter if you are using an r6gd node type. If you don't want data tiering, set `--no-data-tiering-enabled`. For more information, see Data tiering.

CacheParameterGroup

Specify the `default.redis6.x.cluster.on` parameter group or a parameter group derived from `default.redis6.x.cluster.on` to create a 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 Redis OSS version 3.2.6 or 4.0.10.
- 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 Redis OSS (cluster mode disabled) and Redis OSS (cluster mode enabled).

Viewing a Replication Group's Details

- Viewing details for a Redis OSS (Cluster Mode Disabled) with replicas
  - Viewing Details for a Redis OSS (Cluster Mode Disabled) Replication Group (Console)
  - Viewing details for a Redis OSS (Cluster Mode Disabled) replication group (AWS CLI)
  - Viewing Details for a Redis OSS (Cluster Mode Disabled) Replication Group (ElastiCache API)
- Viewing a replication group's details: Redis OSS (Cluster Mode Enabled)
  - Viewing details for a Redis OSS (Cluster Mode Enabled) cluster (Console)
  - Viewing details for a Redis OSS (Cluster Mode Enabled) cluster (AWS CLI)
  - Viewing details for a 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 details for a Redis OSS (Cluster Mode Disabled) with replicas

You can view the details of a 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 Redis OSS (Cluster Mode Disabled) Cluster's Details

Viewing Details for a Redis OSS (Cluster Mode Disabled) Replication Group (Console)

To view the details of a Redis OSS (cluster mode disabled) cluster with replicas using the ElastiCache console, see the topic Viewing details of a Redis OSS (Cluster Mode Disabled) cluster (Console).

Viewing details for a Redis OSS (Cluster Mode Disabled) replication group (AWS CLI)

For an AWS CLI example that displays a Redis OSS (cluster mode disabled) replication group's details, see Viewing a replication group's details (AWS CLI).

Viewing Details for a Redis OSS (Cluster Mode Disabled) Replication Group (ElastiCache API)

For an ElastiCache API example that displays a Redis OSS (cluster mode disabled) replication group's details, see Viewing a replication group's details (ElastiCache API).

Viewing a replication group's details: Redis OSS (Cluster Mode Enabled)

Viewing details for a Redis OSS (Cluster Mode Enabled) cluster (Console)

To view the details of a Redis OSS (cluster mode enabled) cluster using the ElastiCache console, see Viewing details for a Redis OSS (Cluster Mode Enabled) cluster (Console).

Viewing details for a Redis OSS (Cluster Mode Enabled) cluster (AWS CLI)

For an ElastiCache CLI example that displays a Redis OSS (cluster mode enabled) replication group's details, see Viewing a replication group's details (AWS CLI).

Viewing details for a Redis OSS (Cluster Mode Enabled) Cluster (ElastiCache API)

For an ElastiCache API example that displays a 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 Redis OSS (cluster mode disabled) or a Redis OSS (cluster mode enabled) replication group, you will be interested in different endpoints.

Redis OSS (Cluster Mode Disabled)

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 (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 (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.

Redis OSS (Cluster Mode Enabled)

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 Redis OSS (cluster mode disabled) clusters. 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, Redis OSS, behind the scenes, determines 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 Redis OSS (Cluster Mode Disabled) Cluster's Endpoints (Console)
- Finding Endpoints for a Redis OSS (Cluster Mode Enabled) Cluster (Console)
- Finding the Endpoints for Replication Groups (AWS CLI)
- Finding Endpoints for Replication Groups (ElastiCache API)

Modifying a replication group

Important Constraints

- Currently, ElastiCache supports limited modifications of a 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 Redis OSS (cluster mode enabled) cluster with the API operation `ModifyReplicationGroupShardConfiguration` (CLI: `modify-replication-group-shard-configuration`). For more information, see Scaling clusters in Redis OSS (Cluster Mode Enabled).

  Other modifications to a Redis OSS (cluster mode enabled) cluster require that you create a cluster with the new cluster incorporating the changes.
- You can upgrade Redis OSS (cluster mode disabled) and 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 Engine versions and upgrading .
- You can upgrade an existing ElastiCache (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 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 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 Redis OSS (cluster mode enabled) replication group.

Topics

Using the AWS Management Console

To modify a 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 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 Redis OSS clients to connect using both cluster mode enabled and cluster mode disabled. After you migrate all 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 (Redis OSS) User Guide*.

Using the ElastiCache API

The following ElastiCache API operation enables Multi-AZ on an existing 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 (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.

Deleting a Replication Group (Console)

To delete a cluster that has replicas, see Deleting a cluster.

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 Redis OSS replication group using the AWS Management Console, the AWS CLI, or the ElastiCache API. If your replication group is a 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 Redis OSS replication group, choose the operation from the following table that fits your situation.

To Do This	For Redis OSS (cluster mode enabled)	For Redis OSS (cluster mode disabled)
Add replicas	Increasing the number of replicas in a shard	Increasing the number of replicas in a shard Adding a read replica, for Redis OSS (Cluster Mode Disabled) replication groups
Delete replicas	Decreasing the number of replicas in a shard	Decreasing the number of replicas in a shard Deleting a read replica, for Redis OSS (Cluster Mode Disabled) replication groups

Increasing the number of replicas in a shard

You can increase the number of replicas in a Redis OSS (cluster mode enabled) shard 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 Redis OSS (cluster mode enabled) replication group.

To increase the number of replicas in Redis OSS 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 **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 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 Redis OSS (cluster mode disabled) replication groups where there is only one node group 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 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 Redis OSS (cluster mode disabled) replication groups, the shard ID is always `0001`. To find a 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 (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 Redis OSS (cluster mode disabled) group with a single node group or a 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 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 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 Redis OSS (cluster mode disabled) replication groups where there is only one node group, 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 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 Redis OSS (cluster mode disabled) replication groups, the node group (shard) ID is always `0001`. To find a 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 (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 Redis OSS (cluster mode disabled) group with a single node group or a 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 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 Redis OSS (cluster mode enabled), or in a replication group for Redis OSS (cluster mode disabled):

- For 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 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

The following procedure uses the console to decrease the number of replicas in a Redis OSS (cluster mode enabled) replication group.

To decrease the number of replicas in a 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 **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 (Redis OSS) automatically selects replica nodes for deletion. While doing so, ElastiCache (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 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 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 Redis OSS (cluster mode disabled) replication groups, the shard ID is always `0001`. To find a 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 (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 Redis OSS (cluster mode disabled) group with a single node group or a 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 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 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 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 Redis OSS (cluster mode disabled) replication groups, the node group ID is always `0001`. To find a 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 (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 Redis OSS (cluster mode disabled) group with a single node group or a 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 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 Redis OSS (Cluster Mode Disabled) replication groups

Information in the following topic applies to 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 Redis OSS (cluster mode disabled) cluster.

A 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 Redis OSS (cluster mode enabled) replication group, see the following:

- Scaling clusters in Redis OSS (Cluster Mode Enabled)
- Increasing the number of replicas in a shard

You can add a read replica to a Redis OSS (cluster mode disabled) cluster using the ElastiCache Console, the AWS CLI, or the ElastiCache API.

Related topics

- Adding nodes to a 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 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 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 Redis OSS (Cluster Mode Disabled) replication groups

Information in the following topic applies to Redis OSS (cluster mode disabled) replication groups only.

As read traffic on your Redis OSS replication group changes, you might want to add or remove read replicas. Removing a node from a Redis OSS (cluster mode disabled) 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 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 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 Redis OSS replication group, see the following:

- Using the AWS Management Console
- Using the AWS CLI
- Using the ElastiCache API
- Scaling clusters in Redis OSS (Cluster Mode Enabled)
- Decreasing the number of replicas in a shard

Promoting a read replica to primary, for Redis OSS (cluster mode disabled) replication groups

Information in the following topic applies to only Redis OSS (cluster mode disabled) replication groups.

You can promote a 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 Redis OSS (cluster mode disabled) replication group. To promote a 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.

Multi-AZ is not available on replication groups running Redis OSS 2.6.13 or earlier.

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 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 **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 maintenance

Every cluster and replication group has a weekly maintenance window during which any system changes are applied. 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
`ap-northeast-1`	Asia Pacific (Tokyo) Region	13:00–21:00 UTC
`ap-northeast-2`	Asia Pacific (Seoul) Region	12:00–20:00 UTC

Region Code	Region Name	Region Maintenance Window
`ap-northeast-3`	Asia Pacific (Osaka) Region	12:00–20:00 UTC
`ap-southeast-3`	Asia Pacific (Jakarta) Region	14:00–22:00 UTC
`ap-south-1`	Asia Pacific (Mumbai) Region	17:30–1:30 UTC
`ap-southeast-1`	Asia Pacific (Singapore) Region	14:00–22:00 UTC
`cn-north-1`	China (Beijing) Region	14:00–22:00 UTC
`cn-northwest-1`	China (Ningxia) Region	14:00–22:00 UTC
`ap-east-1`	Asia Pacific (Hong Kong) Region	13:00–21:00 UTC
`ap-southeast-2`	Asia Pacific (Sydney) Region	12:00–20:00 UTC
`eu-west-3`	EU (Paris) Region	23:59–07:29 UTC
`af-south-1`	Africa (Cape Town) Region	13:00–21:00 UTC
`eu-central-1`	Europe (Frankfurt) Region	23:00–07:00 UTC
`eu-west-1`	Europe (Ireland) Region	22:00–06:00 UTC
`eu-west-2`	Europe (London) Region	23:00–07:00 UTC
`me-south-1`	Middle East (Bahrain) Region	13:00–21:00 UTC
`me-central-1`	Middle East (UAE) Region	13:00–21:00 UTC
`eu-south-1`	Europe (Milan) Region	21:00–05:00 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-west-1`	AWS GovCloud (US) region	06:00–14:00 UTC

Region Code	Region Name	Region Maintenance Window
`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—Managing node replacement
- Modifying a replication group—Changing a replication group's maintenance window

Configuring engine parameters using 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 parameters, see Redis OSS-specific parameters.

As you would expect, some parameter values, such as `maxmemory`, are determined by the engine and node type. For a table of these parameter values by node type, see Redis OSS node-type specific parameters.

Topics

Parameter management

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.
- 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 3.2.10, you can only use parameter groups, default or custom, from the Redis OSS 3.2 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.
- 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 Redis OSS-specific parameters.

  For more information, see Rebooting nodes.

  Redis OSS (Cluster Mode Enabled) parameter changes

  If you make changes to the following parameters on a 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 Redis OSS (cluster mode enabled) 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.

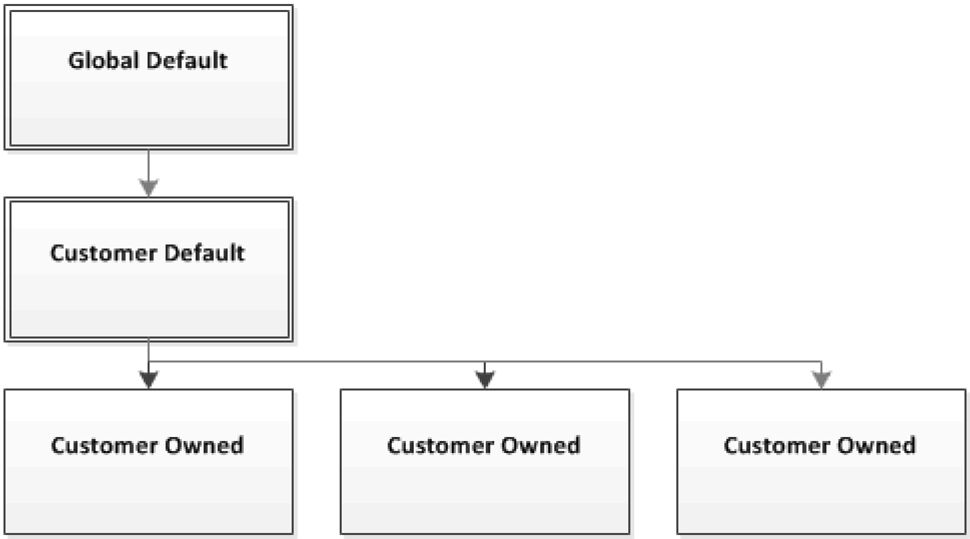
Changes to other parameters do not require this.

- You can associate parameter groups with 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

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, see [Redis OSS-specific parameters](#).

Creating a 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 a 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 *redis3.2*, 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 a parameter group.

Creating a 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 *myRed28* using the redis2.8 family as the template.

For Linux, macOS, or Unix:

```
aws elasticache create-cache-parameter-group \
    --cache-parameter-group-name myRed28  \
    --cache-parameter-group-family redis2.8 \
    --description "My first parameter group"
```

For Windows:

```
aws elasticache create-cache-parameter-group ^
    --cache-parameter-group-name myRed28  ^
    --cache-parameter-group-family redis2.8 ^
    --description "My first parameter group"
```

The output from this command should look something like this.

```
{
    "CacheParameterGroup": {
        "CacheParameterGroupName": "myRed28",
        "CacheParameterGroupFamily": "redis2.8",
        "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 a parameter group](#).

For more information, see `create-cache-parameter-group`.

Creating a 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, `redis2.8`.
- `Description` — A user supplied description for the parameter group.

Example

The following example creates a parameter group named *myRed28* using the redis2.8 family as the template.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=CreateCacheParameterGroup
   &CacheParameterGroupFamily=redis2.8
   &CacheParameterGroupName=myRed28
   &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>myRed28</CacheParameterGroupName>
      <CacheParameterGroupFamily>redis2.8</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 a parameter group.

For more information, see `CreateCacheParameterGroup`.

Listing 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 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 *myRed28*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameter-groups \
    --cache-parameter-group-name myRed28
```

For Windows:

```
aws elasticache describe-cache-parameter-groups ^
    --cache-parameter-group-name myRed28
```

The output of this command will look something like this, listing the name, family, and description for the parameter group.

```
{
    "CacheParameterGroups": [
     {
         "CacheParameterGroupName": "myRed28",
         "CacheParameterGroupFamily": "redis2.8",
         "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-redis32",
            "CacheParameterGroupFamily": "redis3.2",
            "Description": "custom parameter group with reserved-memory > 0"
        },
        {
            "CacheParameterGroupName": "default.memcached1.4",
            "CacheParameterGroupFamily": "memcached1.4",
            "Description": "Default parameter group for memcached1.4"
        },
        {
            "CacheParameterGroupName": "default.redis2.6",
            "CacheParameterGroupFamily": "redis2.6",
            "Description": "Default parameter group for redis2.6"
        },
        {
            "CacheParameterGroupName": "default.redis2.8",
            "CacheParameterGroupFamily": "redis2.8",
            "Description": "Default parameter group for redis2.8"
        },
        {
            "CacheParameterGroupName": "default.redis3.2",
            "CacheParameterGroupFamily": "redis3.2",
            "Description": "Default parameter group for redis3.2"
        },
        {
            "CacheParameterGroupName": "default.redis3.2.cluster.on",
            "CacheParameterGroupFamily": "redis3.2",
            "Description": "Customized default parameter group for redis3.2 with
 cluster mode on"
        },
```

```
        {
            "CacheParameterGroupName": "default.redis5.6.cluster.on",
            "CacheParameterGroupFamily": "redis5.0",
            "Description": "Customized default parameter group for redis5.6 with
 cluster mode on",
            "isGlobal": "yes"
        },
    ]
}
```

For more information, see describe-cache-parameter-groups.

Listing 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 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>myRedis28</CacheParameterGroupName>
        <CacheParameterGroupFamily>redis2.8</CacheParameterGroupFamily>
        <Description>My custom Redis 2.8 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 *myRed28*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameterGroups
   &CacheParameterGroupName=myRed28
   &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>myRed28</CacheParameterGroupName>
        <CacheParameterGroupFamily>redis2.8</CacheParameterGroupFamily>
        <Description>My custom Redis 2.8 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 a 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 a 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 parameters and their values for the parameter group *myRedis28*.

For Linux, macOS, or Unix:

```
aws elasticache describe-cache-parameters \
    --cache-parameter-group-name myRedis28
```

For Windows:

```
aws elasticache describe-cache-parameters ^
```

```
    --cache-parameter-group-name myRed28
```

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 parameters for the parameter group *myRed28*.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DescribeCacheParameters
   &CacheParameterGroupName=myRed28
   &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>1000</Value>
            <CacheClusterClass>cache.c1.medium</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>6000</Value>
            <CacheClusterClass>cache.c1.xlarge</CacheClusterClass>
```

```
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>7100</Value>
            <CacheClusterClass>cache.m1.large</CacheClusterClass>
          </CacheNodeTypeSpecificValue>
          <CacheNodeTypeSpecificValue>
            <Value>1300</Value>
            <CacheClusterClass>cache.m1.small</CacheClusterClass>
          </CacheNodeTypeSpecificValue>

...output omitted...

    </CacheClusterClassSpecificParameters>
  </DescribeCacheParametersResult>
  <ResponseMetadata>
    <RequestId>6d355589-af49-11e0-97f9-279771c4477e</RequestId>
  </ResponseMetadata>
</DescribeCacheParametersResponse>
```

For more information, see `DescribeCacheParameters`.

Modifying a 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 Redis OSS-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. Choose **Save Changes**.

6. To find the name of the parameter you changed, see Redis OSS-specific parameters. If you have a 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.

   > **Redis OSS (Cluster Mode Enabled) parameter changes**
   >
   > If you make changes to the following parameters on a 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 Redis OSS (cluster mode enabled) 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

To find the name and permitted values of the parameter you want to change, see [Redis OSS-specific parameters](#)

The following sample code sets the value of two parameters, *reserved-memory-percent* and *cluster-enabled* on the parameter group `myredis32-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 Redis OSS (cluster mode enabled) clusters (replication groups).

For Linux, macOS, or Unix:

```
aws elasticache modify-cache-parameter-group \
    --cache-parameter-group-name myredis32-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 myredis32-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-redis32-on-30"
}
```

For more information, see [`modify-cache-parameter-group`](#).

To find the name of the parameter you changed, see [Redis OSS-specific parameters](#).

If you have a 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.

> **Redis OSS (Cluster Mode Enabled) parameter changes**
>
> If you make changes to the following parameters on a 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 Redis OSS (cluster mode enabled) 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

To find the name and permitted values of the parameter you want to change, see Redis OSS-specific parameters

The following sample code sets the value of two parameters, *reserved-memory-percent* and *cluster-enabled* on the parameter group `myredis32-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 Redis OSS (cluster mode enabled) clusters (replication groups).

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=ModifyCacheParameterGroup
   &CacheParameterGroupName=myredis32-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 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.

Redis OSS (Cluster Mode Enabled) parameter changes

If you make changes to the following parameters on a 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 Redis OSS (cluster mode enabled) cluster. See Deleting a cluster.
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 a 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 myRed28
```

For Windows:

```
aws elasticache delete-cache-parameter-group ^
    --cache-parameter-group-name myRed28
```

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

The following sample code deletes the *myRed28* parameter group.

```
https://elasticache.us-west-2.amazonaws.com/
   ?Action=DeleteCacheParameterGroup
   &CacheParameterGroupName=myRed28
   &SignatureVersion=4
   &SignatureMethod=HmacSHA256
   &Timestamp=20150202T192317Z
   &Version=2015-02-02
   &X-Amz-Credential=<credential>
```

For more information, see `DeleteCacheParameterGroup`.

Memcached specific parameters

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 a parameter group.

Topics

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 Type: boolean Modifiable: Yes Allowed_Values: 0,1 Changes Take Effect: At launch	An alias for disabling `slab_reassign` , `lru_maintainer_thread` , `lru_segmented` , and `maxconns_fast` commands . When using Memcached 1.5 and higher, `no_modern` also sets the hash_algorithm to `jenkins`. In addition, when using Memcached 1.5.10, `inline_ascii_reponse` is controlled by the parameter `parallelly` . This means that if `no_modern` is disabled then `inline_ascii_reponse` is disabled. From Memcached engine 1.5.16 onward the `inline_ascii_response` parameter no longer applies, so `no_modern` being abled or disabled has no effect on `inline_ascii_reponse` . If `no_modern` is disabled, then `slab_reassign` , `lru_maintainer_thread` , `lru_segmented` , and `maxconns_fast` WILL be enabled. Since `slab_automove` and

Name	Details	Description
		`hash_algorithm` parameters are not SWITCH parameters, their setting is based on the configura tions in the parameter group. 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. **Note** The default configuration value for this parameter 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. Ex isting 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.

Name	Details	Description
`inline_ascii_resp`	Default: 0 Type: boolean Modifiable: Yes Allowed_Values: 0,1 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 Type: boolean Modifiable: Yes Allowed_Values: 0,1 Changes Take Effect: At launch	No longer supported in this version.
`modern`	Default: 1 Type: boolean Modifiable: Yes (requires re-launch if set to `no_modern`) Allowed_Values: 0,1 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 Type: boolean Modifiable: Yes 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`, `lru_crawler`, `lru_maintainer`, `maxconns_fast`, and `hash_algorithm=murmur3`.
`watch`	Default: enabled Type: boolean Modifiable: Yes Changes Take Effect: Immediately Logs can get dropped if user hits their `watcher_logbuf_size` and `worker_logbuf_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.

Name	Details	Description
idle_timeout	Default: 0 (disabled) Type: integer Modifiable: Yes 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 Type: boolean Modifiable: Yes Changes Take Effect: At	Shows the sizes each slab group has consumed. Enabling `track_sizes` lets you run `stats sizes` without the need to run `stats sizes_enable`.
watcher_logbuf_size	Default: 256 (KB) Type: integer Modifiable: Yes 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 losses.

Name	Details	Description
`worker_logbuf_size`	Default: 64 (KB) Type: integer Modifiable: Yes 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) Type: integer Modifiable: Yes Changes Take Effect: At	Specifies the maximum size of a slab. Setting smaller slab size uses memory more efficiently. Items larger than `slab_chunk_max` are split over multiple slabs.
`lru_crawler metadump [all|1|2|3]`	Default: disabled Type: boolean Modifiable: Yes Changes Take Effect: Im ly	if lru_crawler is enabled this command dumps all keys. `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)	

Name	Details	Description
	Type: boolean Modifiable: Yes 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. Values: 0, 1 (user can do a `flush_all` when the value is 0).
`hash_algorithm`	Default: jenkins Type: string Modifiable: Yes Changes Take Effect: At launch	The hash algorithm to be used. Permitted values: murmur3 and jenkins.

Name	Details	Description
`lru_crawler`	Default: 0 (disabled) Type: boolean Modifiable: Yes Changes Take Effect: Af **Note** 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. To temporarily enable, run `lru_crawler enable` at the command line. `lru_crawler 1,3,5` crawls slab classes 1, 3, and 5 looking for expired items to add to the freelist. Values: 0,1 **Note** 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 a parameter group.

Name	Details	Description
`lru_maintainer`	Default: 0 (disabled) Type: boolean Modifiable: Yes Changes Take Effect: At launch	A background thread that shuffles items between the LRUs as capaci ties are reached. Values: 0, 1.
`expirezero_does_no t_evict`	Default: 0 (disabled) Type: boolean Modifiable: Yes Changes Take Effect: At launch	When used with `lru_maint ainer` , makes items with an expiration time of 0 unevictable. **Warning** This can crowd out memory available for other evictable items. 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 wh en 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 Type: integer	The backlog queue limit.

Name	Details	Description
	Modifiable: No	
`binding_protocol`	Default: auto Type: string Modifiable: Yes Changes Take Effect: After resta	The binding protocol. Permissible values are: `ascii` and `auto`. For guidance on modifying the value of `binding_protocol` , see [Modifying a parameter group]().
`cas_disabled`	Default: 0 (false) Type: Boolean Modifiable: Yes 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 Type: integer Modifiable: Yes 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 Type: float Modifiable: Yes 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) Type: Boolean Modifiable: Yes 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) Type: Boolean Modifiable: No	If 1 (true), ElastiCache will try to use large memory pages.
lock_down_paged_memory	Default: 0 (false) Type: Boolean Modifiable: No	If 1 (true), ElastiCache will lock down all paged memory.
max_item_size	Default: 1048576 Type: integer Modifiable: Yes 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 Type: integer Modifiable: No	The maximum number of simultaneous connections.
maximize_core_file_limit	Default: 0 (false) Type: Boolean Modifiable: Changes Take Effect: After resta	If 1 (true), ElastiCache will maximize the core file limit.
memcached_connections_overhead	Default: 100 Type: integer Modifiable: Yes 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 Type: integer 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.m1.small` has a `max_cache_memory` of 1300MB. With the default `memcached_connections_overhead` value of 100MB, the Memcached process will have 1200MB 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.

> **Important**
>
> For the `cache.t1.micro` node type, the value for `memcached_connections_overhead` is determined as follows:
>
> - If you cluster is using the default parameter group, ElastiCache will set the value for `memcached_connections_overhead` to 13MB.

- If your cluster is using a parameter group that you have created yourself, you can set the value of `memcached_connections_overhead` to a value of your choice.

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.t1.micro	213	1
cache.t2.micro	555	1
cache.t2.small	1588	1
cache.t2.medium	3301	2
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.m1.small	1301	1
cache.m1.medium	3350	1
cache.m1.large	7100	2
cache.m1.xlarge	14600	4

Node type	max_cache_memory (in megabytes)	num_threads
cache.m2.xlarge	33800	2
cache.m2.2xlarge	30412	4
cache.m2.4xlarge	68000	16
cache.m3.medium	2850	1
cache.m3.large	6200	2
cache.m3.xlarge	13600	4
cache.m3.2xlarge	28600	8
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

Node type	max_cache_memory (in megabytes)	num_threads
cache.m6g.2xlarge	26671	8
cache.m6g.4xlarge	53516	16
cache.m6g.8xlarge	107000	32
cache.m6g.12xlarge	160900	48
cache.m6g.16xlarge	214577	64
cache.c1.xlarge	6600	8
cache.r3.large	13800	2
cache.r3.xlarge	29100	4
cache.r3.2xlarge	59600	8
cache.r3.4xlarge	120600	16
cache.r3.8xlarge	120600	32
cache.r4.large	12590	2
cache.r4.xlarge	25652	4
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

Node type	max_cache_memory (in megabytes)	num_threads
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.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

Note

All T2 instances are created in an Amazon Virtual Private Cloud (Amazon VPC).

Redis OSS-specific parameters

If you do not specify a parameter group for your 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 a parameter group.

Topics

Redis OSS 7 parameter changes

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.

Parameters added in Redis OSS 7 are as follows.

Name	Details	Description
`cluster-allow-pubsubshard-when-down`	Permitted values: `yes`, no Default: `yes` Type: string Modifiable: Yes 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.
`cluster-preferred-endpoint-type`	Permitted values: `ip`, `tls-dynamic` Default: `tls-dynamic` Type: string Modifiable: Yes 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 Default: no Type: string	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.

Name	Details	Description
	Modifiable: Yes Changes take effect: Immediately across all nodes in the cluster.	
`hash-max-listpack-entries`	Permitted values: `0+` Default: 512 Type: integer Modifiable: Yes Changes take effect: Immediately across all nodes in the cluster.	The maximum number of hash entries in order for the dataset to be compressed.
`hash-max-listpack-value`	Permitted values: `0+` Default: 64 Type: integer Modifiable: Yes Changes take effect: Immediately across all nodes in the cluster.	The threshold of biggest hash entries in order for the dataset to be compressed.

Name	Details	Description
`zset-max-listpack-entries`	Permitted values: `0+` Default: 128 Type: integer Modifiable: Yes 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+` Default: 64 Type: integer Modifiable: Yes 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+` Default: 512 Type: integer Modifiable: Yes 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+` Default: 64 Type: integer Modifiable: Yes Changes take effect: Immediately across all nodes in the cluster.	Use `listpack` instead of `ziplist` for representing small hash encoding
`zset-max-ziplist-entries`	Permitted values: `0+` Default: 128 Type: integer Modifiable: Yes 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-value`	Permitted values: `0+` Default: `64` Type: integer Modifiable: Yes 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: Default: `-2` Type: integer Modifiable: Yes 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 Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis6.x.cluster.on` – Use this parameter group, or one derived from it, for 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, only *noeviction*, *volatile-lru* and *allkeys-lru* max-memory policies are supported with r6gd node types.

For more information, see ElastiCache (Redis OSS) version 6.2 (enhanced) and ElastiCache (Redis OSS) version 6.0 (enhanced).

Parameters added in Redis OSS 6.x are as follows.

Name	Details	Description
`acl-pubsub-default (added in 6.2)`	Permitted values: `resetchannels , allchannels` Default: `allchannels` Type: string Modifiable: Yes Changes take effect: The existing Redis OSS users associated to the cluster will continue to have existing permissions. Either update the users or reboot the cluster to update the existing Redis OSS users.	Default pubsub channel permissions for ACL users deployed to this cluster.
`cluster-allow-reads-when-down (added in 6.0)`	Default: no Type: string Modifiable: Yes	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.

Name	Details	Description
	Changes take effect: Immediately across all nodes in the cluster	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 Type: number Modifiable: Yes 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. When the tracked key is modified, invalidat ion 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.
`acllog-max-len` (added in 6.0)	Default: 128 Type: number Modifiable: Yes Changes take effect: Immediately across all nodes in the cluster	This value corresponds to the max number of entries in the ACL log.

Name	Details	Description
`active-expire-effort (added in 6.0)`	Default: 1 Type: number Modifiable: Yes 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. 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.
`lazyfree-lazy-user-del (added in 6.0)`	Default: no Type: string Modifiable: Yes 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 Default: yes Type: boolean Modifiable: Yes Changes take effect: Immediately	Always enable Lua effect replication or not in Lua scripts

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 Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis5.0.cluster.on` – Use this parameter group, or one derived from it, for Redis OSS (cluster mode enabled) clusters and replication groups.

Parameters added in Redis OSS 5.0.3

Name	Details	Description
`rename-commands`	Default: none Type: string Modifiable: Yes 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: `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`

Name	Details	Description
		EXPIREAT FLUSHALL FLUSHDB GEOADD GEOHASH GEOPOS GEODIST GEORADIUS GEORADIUSBYMEMBER GET GETBIT GETRANGE GETSET HDEL HEXISTS HGET HGETALL HINCRBY HINCRBYFL OAT HKEYS HLEN HMGET HMSET HSET HSETNX HSTRLEN HVALS INCR INCRBY INCRBYFLOAT INFO KEYS LASTSAVE LINDEX LINSERT LLEN LPOP LPU SH 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 ZINTERSTO RE ZLEXCOUNT ZPOPMAX ZPOPMIN ZRANGE ZRANGEBYLEX ZREVRANGE BYLEX ZRANGEBYSCORE ZRANK ZREM ZREMRANGEBYLEX ZREMRANGEBYRANK ZREMRANGEBYSCORE ZREVRANGE ZREVRANGEBYSCORE ZREVRANK ZSCORE ZUNIONSTORE SCAN SSCAN HSCAN

Name	Details	Description
		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 (Redis OSS) version 5.0.6 (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 Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis5.0.cluster.on` – Use this parameter group, or one derived from it, for 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+ Default: 4096 Type: integer Modifiable: Yes 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+	The stream data structure is a radix tree of nodes that encode multiple items inside. Use this configuration to specify the maximum

Name	Details	Description
	Default: 100 Type: integer Modifiable: Yes Changes take effect: Immediately	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
`active-defrag-max-scan-fields`	Permitted values: 1 to 1000000 Default: 1000 Type: integer Modifiable: Yes 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 Default: yes Type: boolean Modifiable: Yes Changes take effect: Immediately	Always enable Lua effect replication or not in Lua scripts
`replica-ignore-maxmemory`	Default: yes Type: boolean Modifiable: No	Determines if replica ignores `maxmemory` setting by not evicting items independent from the primary

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 Type: boolean Modifiable: No 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 Type: integer Modifiable: No 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-soft-limit`	Default: For values see Redis OSS node-type specific parameters Type: integer Modifiable: No Former name: client-output-buffer-limit-slave-soft-limit	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-replica-soft-seconds` .
`client-output-buffer-limit-`	Default: 60 Type: integer	For Redis OSS read replicas: If a client's output buffer remains at `client-output-buffer-limit-replica-soft-limit`

Name	Details	Description
`replica-soft-seconds`	Modifiable: No Former name: client-output-buffer-limit-slave-soft-seconds	bytes for longer than this number of seconds, the client will be disconnected.
`replica-allow-chaining`	Default: no Type: string Modifiable: No 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 Type: integer Modifiable: Yes Former name: min-slaves-to-write 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. 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 Type: integer Modifiable: Yes Former name: min-slaves-max-lag 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. 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 Type: boolean Modifiable: Yes Former name: close-on-slave-write 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 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 Redis OSS (cluster mode disabled) clusters and replication groups.
- `default.redis4.0.cluster.on` – Use this parameter group, or one derived from it, for 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` Default: volatile-lru Type: string Modifiable: Yes 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 using 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 Default: no Type: boolean Modifiable: Yes Changes take place: immediately	Performs an asynchronous delete on evictions.
`lazyfree-lazy-expire`	Permitted values: yes/no Default: no Type: boolean Modifiable: Yes Changes take place: immediately	Performs an asynchronous delete on expired keys.
`lazyfree-lazy-server-del`	Permitted values: yes/no Default: no Type: boolean Modifiable: Yes Changes take place: immediately	Performs an asynchronous delete for commands which update values.

Name	Details	Description
`slave-lazy-flush`	Permitted values: N/A Default: no Type: boolean Modifiable: No Changes take place: N/A	Performs an asynchronous flushDB during slave sync.
LFU parameters		
`lfu-log-factor`	Permitted values: any integer > 0 Default: 10 Type: integer Modifiable: Yes 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 Default: 1 Type: integer Modifiable: Yes 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 Default: no Type: boolean Modifiable: Yes Changes take place: immediately	Enabled active defragmen tation.
`active-defrag-ignore-bytes`	Permitted values: 10485760-104857600 Default: 104857600 Type: integer Modifiable: Yes Changes take place: immediately	Minimum amount of fragmentation waste to start active defrag.
`active-defrag-threshold-lower`	Permitted values: 1-100 Default: 10 Type: integer Modifiable: Yes Changes take place: immediately	Minimum percentage of fragmentation to start active defrag.

Name	Details	Description
`active-defrag-threshold-upper`	Permitted values: 1-100 Default: 100 Type: integer Modifiable: Yes Changes take place: immediately	Maximum percentage of fragmentation at which we use maximum effort.
`active-defrag-cycle-min`	Permitted values: 1-75 Default: 25 Type: integer Modifiable: Yes Changes take place: immediately	Minimal effort for defrag in CPU percentage.
`active-defrag-cycle-max`	Permitted values: 1-75 Default: 75 Type: integer Modifiable: Yes Changes take place: immediately	Maximal effort for defrag in CPU percentage.

Client output buffer parameters

Name	Details	Description
`client-query-buffer-limit`	Permitted values: 1048576-1073741824 Default: 1073741824 Type: integer Modifiable: Yes Changes take place: immediately	Max size of a single client query buffer.
`proto-max-bulk-len`	Permitted values: 1048576-536870912 Default: 536870912 Type: integer Modifiable: Yes Changes take place: immediately	Max size of a single element request.

Redis OSS 3.2.10 parameter changes

Parameter group family: redis3.2

ElastiCache (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 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 Redis OSS (cluster mode enabled) replication group.

Topics

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 Type: integer Modifiable: No	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 elements. For a fixed maximum size, use -5 through -1, meaning: • -5: max size: 64 Kb - not recommended for normal workloads • -4: max size: 32 Kb - not recommended • -3: max size: 16 Kb - not recommended • -2: max size: 8 Kb - recommended • -1: max size: 4 Kb - recommended

Name	Details	Description
		• Positive numbers mean store up to exactly that number of elements per list node.
`list-compress-depth`	Default: 0 Type: integer Modifiable: Yes 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: • 0: Disable all compression. • 1: Start compressing with the 1st node in from the head and tail. [head]->node->node->...->node->[tail] All nodes except [head] and [tail] compress. • 2: Start compressing with the 2nd node in from the head and tail. [head]->[next]->node->node->...->node->[prev]->[tail] [head], [next], [prev], [tail] do not compress. All other nodes compress. • Etc.

Name	Details	Description
`cluster-enabled`	Default: no/yes * Type: string Modifiable: No	Indicates whether this is a Redis OSS (cluster mode enabled) replication group in cluster mode (yes) or a Redis OSS (cluster mode enabled) replication group in non-cluster mode (no). Redis OSS (cluster mode enabled) replication groups in cluster mode can partition their data across up to 500 node groups. * Redis OSS 3.2.*x* has two default parameter groups. • `default.redis3.2` – default value no. • `default.redis3.2.cluster.on` – default value `yes`. .
`cluster-require-full-coverage`	Default: no Type: boolean Modifiable: yes Changes Take Effect: Immediately	When set to `yes`, 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. 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 Type: integer Modifiable: Yes 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. A value greater than 16000 is not recommended, because at that point the dense representation is more memory efficient. 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.
`reserved-memory-percent`	Default: 25 Type: integer Modifiable: Yes 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. This parameter is specific to ElastiCache, and is not part of the standard Redis OSS distribution. For more information, see `reserved-memory` and [Managing Reserved Memory]().

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 cache clusters. Otherwise, no.	Modifiable was No.
`databases`	Modifiable: Yes if the parameter group is not associated with any cache clusters. Otherwise, no.	Modifiable was No.
`appendonly`	Default: off Modifiable: No	If you want to upgrade from an earlier Redis OSS version, you must first turn `appendonly` off.
`appendfsync`	Default: off Modifiable: No	If you want to upgrade from an earlier Redis OSS version, you must first turn `appendfsync` off.
`repl-timeout`	Default: 60 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

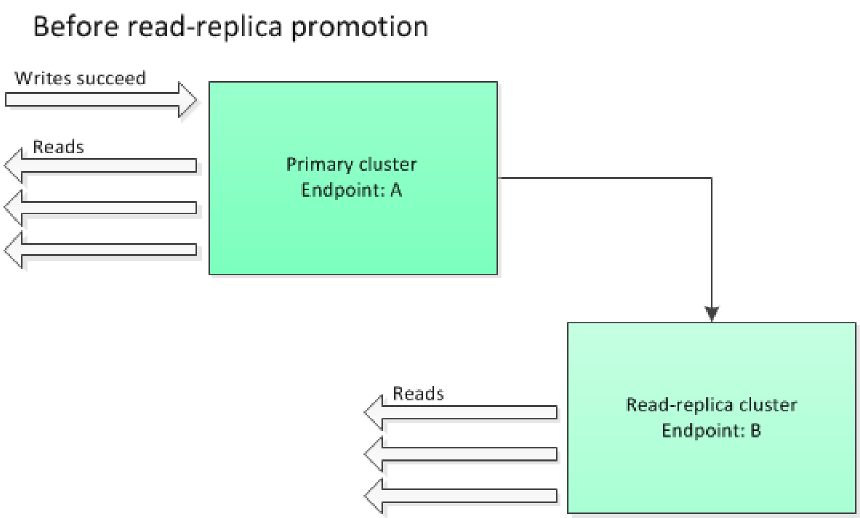
Parameter group family: redis2.8

For Redis OSS 2.8.23 the following additional parameter is supported.

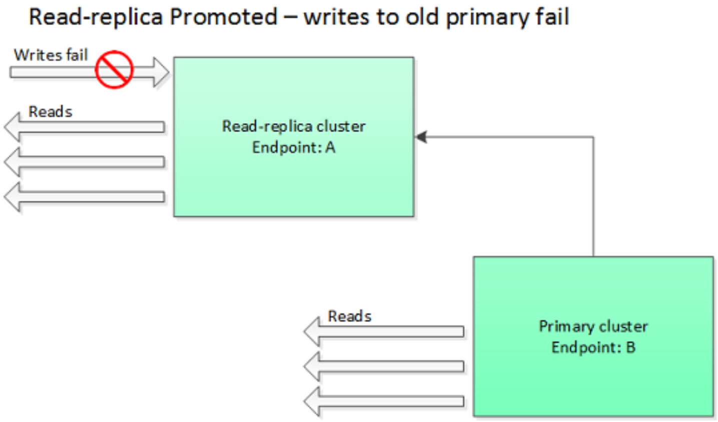
Name	Details	Description
`close-on-slave-write`	Default: yes Type: string (yes/no) Modifiable: Yes Changes Take Effect: Immediately	If enabled, clients who attempt to write to a read-only replica will be disconnected.

How close-on-slave-write works

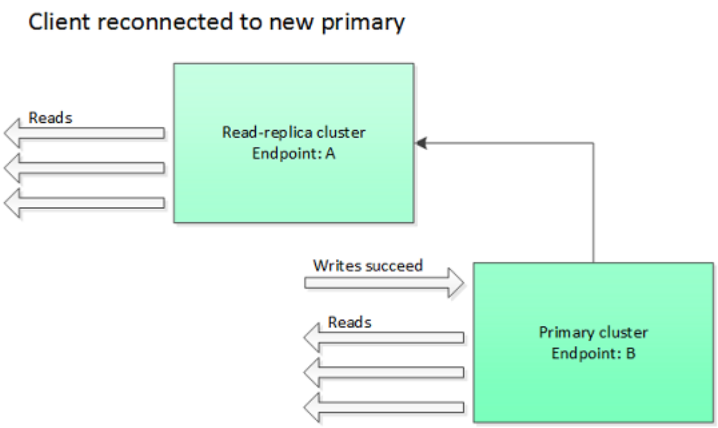
The `close-on-slave-write` parameter is introduced by Amazon ElastiCache to give you more control over how your cluster responds when a primary node and a read replica node swap roles due to promoting a read replica to primary.



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 Type: integer Modifiable: Yes 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-slaves-to-write, then the primary will stop accepting writes at that point. If either this parameter or min-slaves-to-write is 0, then the primary node will always accept writes requests, even if no replicas are available.
`min-slaves-to-write`	Default: 0 Type: integer Modifiable: Yes 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. If either this parameter or min-slaves-max-lag is 0, then the primary node will always accept writes requests, even if no replicas are available.

Name	Details	Description
`notify-keyspace-events`	Default: (an empty string) Type: string Modifiable: Yes Changes Take Effect: Im ly	The types of keyspace events that Redis OSS can notify clients of. Each event type is represented by a single letter: • **K** — Keyspace events, published with a prefix of *__keyspace@<db>__* • **E** — Key-event events, published with a prefix of *__keyevent@<db>__* • **g** — Generic, non-specific commands such as *DEL*, *EXPIRE*, *RENAME*, etc. • **$** — String commands • **l** — List commands • **s** — Set commands • **h** — Hash commands • **z** — Sorted set commands • **x** — Expired events (events generated every time a key exp ires) • **e** — Evicted events (events generated when a key is evicted for maxmemory) •

Name	Details	Description
		A — An alias for *g$lshzxe* You can have any combination of these event types. For example, *AKE* means that Redis OSS can publish notifications of all event types. Do not use any characters other than those listed above; attempts to do so will result in error messages. 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 Type: integer Modifiable: Yes 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 from the backlog to catch up with the primary node. If the `psync` fails, then a full sync is required. The minimum value for this parameter is 16384. **Note** 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 Type: integer Modifiable: Yes 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. If this parameter is set to 0, then the backlog buffer will never be released.
`repl-timeout`	Default: 60 Type: integer Modifiable: Yes Changes Take Effect: Im ly	Represents the timeout period, in seconds, for: • Bulk data transfer during synchronization, from the read replica's perspective • Primary node timeout from the replica's perspective • 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 Type: string (yes/no) Modifiable: Yes Changes take place: 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. 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 Type: string Modifiable: Yes 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. The default value is *no*, meaning AOF is turned off. Set this parameter to *yes* to enable AOF. For more information, see Mitigating Failures. **Note** Append Only Files (AOF) is not supported for cache.t1.micro and cache.t2.* nodes. For nodes of this type, the `appendonly` parameter value is ignored.

Name	Details	Description
		Note For Multi-AZ replication groups, AOF is not allowed.
`appendfsync`	Default: everysec Type: string Modifiable: Yes Changes Take Effect: Immediately	When `appendonly` is set to yes, controls how often the AOF output buffer is written to disk: • *no* — the buffer is flushed to disk on an as-needed basis. • *everysec* — the buffer is flushed once per second. This is the default. • *always* — the buffer is flushed every time that data in the cluster is modified. • Appendfsync is not supported for versions 2.8.22 and later.
`client-output-buffer-limit-normal-hard-limit`	Default: 0 Type: integer Modifiable: Yes 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).

Name	Details	Description
`client-output-buffer-limit-normal-soft-limit`	Default: 0 Type: integer Modifiable: Yes 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 Type: integer Modifiable: Yes 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).
`client-output-buffer-limit-pubsub-hard-limit`	Default: 33554432 Type: integer Modifiable: Yes 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 Type: integer Modifiable: Yes 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`.

Name	Details	Description
`client-output-buffer-limit-pubsub-soft-seconds`	Default: 60 Type: integer Modifiable: Yes 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] Type: integer Modifiable: No	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-slave-soft-limit`	Default: For values see [Redis OSS node-type specific parameters] Type: integer 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 Type: integer 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.

Name	Details	Description
`databases`	Default: 16 Type: integer Modifiable: No Changes take place: At Creation	The number of logical partitions the databases is split into. We recommend keeping this value low. 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 Type: integer Modifiable: Yes 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.
`hash-max-ziplist-value`	Default: 64 Type: integer Modifiable: Yes 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 Type: integer Modifiable: Yes 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.

Name	Details	Description
`list-max-ziplist-value`	Default: 64 Type: integer Modifiable: Yes 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 Type: integer Modifiable: No	The maximum execution time for a Lua script, in milliseconds, before ElastiCache takes action to stop the script. 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.
`maxclients` This value applies to all instance types except those explicity specified	Default: 65000 Type: integer Modifiable: No t2.medium Default: 20000 Type: integer Modifiable: No t2.small Default: 20000 Type: integer Modifiable: No	The maximum number of clients that can be connected at one time.

Name	Details	Description
	t2.micro Default: 20000 Type: integer Modifiable: No	
	t4g.micro Default: 20000 Type: integer Modifiable: No	
	t3.medium Default: 46000 Type: integer Modifiable: No	
	t3.small Default: 46000 Type: integer Modifiable: No	
	t3.micro Default: 20000 Type: integer Modifiable: No	
`maxmemory-policy`	Default: volatile-lru Type: string Modifiable: Yes Changes Take Effect: Immediately	The eviction policy for keys when maximum memory usage is reached. Valid values are: `volatile-lru` | `allkeys-lru` | `volatile-random` | `allkeys-random` | `volatile-ttl` | `noeviction` For more information, see [Using Redis OSS as an LRU cache]().

Name	Details	Description
`maxmemory-samples`	Default: 3 Type: integer Modifiable: Yes 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 Type: integer Modifiable: Yes 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. This parameter is specific to ElastiCache, and is not part of the standard Redis OSS distribution. For more information, see `reserved-memory-percent` and [Managing Reserved Memory](#).
`set-max-intset-entries`	Default: 512 Type: integer Modifiable: Yes 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 Type: string Modifiable: No	Determines whether a read replica in Redis OSS can have read replicas of its own.
`slowlog-log-slower-than`	Default: 10000 Type: integer Modifiable: Yes 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 Type: integer Modifiable: Yes Changes Take Effect: Immediately	The maximum length of the Redis OSS Slow Log.
`tcp-keepalive`	Default: 0 Type: integer Modifiable: Yes 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. **Important** 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 Type: integer Modifiable: Yes Changes Take Effect: Immediately	The number of seconds a node waits before timing out. Values are: • `0` – never disconnect an idle client. • `1-19` – invalid values. • `>=20` – the number of seconds a node waits before disconnecting an idle client.
`zset-max-ziplist-entries`	Default: 128 Type: integer Modifiable: Yes 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 Type: integer Modifiable: Yes 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.

Redis OSS 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 `maxmemory`, `client-output-buffer-limit-slave-hard-limit`, and `client-output-buffer-limit-slave-soft-limit` parameters for each node type. The value of `maxmemory` is the maximum number of bytes available to you for use, data and other uses, on the node. For more information, see [Available memory](#).

> **Note**
>
> The `maxmemory` parameter cannot be modified.

Node type	Maxmemory	Client-output-buffer-limit-slave-hard-limit	Client-output-buffer-limit-slave-soft-limit
cache.t1.micro	142606336	14260633	14260633
cache.t2.micro	581959680	58195968	58195968
cache.t2.small	1665138688	166513868	166513868
cache.t2.medium	3461349376	346134937	346134937
cache.t3.micro	536870912	53687091	53687091
cache.t3.small	1471026299	147102629	147102629
cache.t3.medium	3317862236	331786223	331786223
cache.t4g.micro	536870912	53687091	53687091
cache.t4g.small	1471026299	147102629	147102629
cache.t4g.medium	3317862236	331786223	331786223
cache.m1.small	943718400	94371840	94371840

Node type	Maxmemory	Client-output-buffer-limit-slave-hard-limit	Client-output-buffer-limit-slave-soft-limit
cache.m1.medium	3093299200	309329920	309329920
cache.m1.large	7025459200	702545920	702545920
cache.m1.xlarge	14889779200	1488977920	1488977920
cache.m2.xlarge	17091788800	1709178880	1709178880
cache.m2.2xlarge	35022438400	3502243840	3502243840
cache.m2.4xlarge	70883737600	7088373760	7088373760
cache.m3.medium	2988441600	309329920	309329920
cache.m3.large	6501171200	650117120	650117120
cache.m3.xlarge	14260633600	1426063360	1426063360
cache.m3.2xlarge	29989273600	2998927360	2998927360
cache.m4.large	6892593152	689259315	689259315
cache.m4.xlarge	15328501760	1532850176	1532850176
cache.m4.2xlarge	31889126359	3188912636	3188912636
cache.m4.4xlarge	65257290629	6525729063	6525729063
cache.m4.10xlarge	166047614239	16604761424	16604761424
cache.m5.large	6854542746	685454275	685454275
cache.m5.xlarge	13891921715	1389192172	1389192172
cache.m5.2xlarge	27966669210	2796666921	2796666921
cache.m5.4xlarge	56116178125	5611617812	5611617812

Node type	Maxmemory	Client-output-buffer-limit-slave-hard-limit	Client-output-buffer-limit-slave-soft-limit
cache.m5.12xlarge	168715971994	16871597199	16871597199
cache.m5.24xlarge	337500562842	33750056284	33750056284
cache.m6g.large	6854542746	685454275	685454275
cache.m6g.xlarge	13891921715	1389192172	1389192172
cache.m6g.2xlarge	27966669210	2796666921	2796666921
cache.m6g.4xlarge	56116178125	5611617812	5611617812
cache.m6g.8xlarge	111325552312	11132555231	11132555231
cache.m6g.12xlarge	168715971994	16871597199	16871597199
cache.m6g.16xlarge	225000375228	22500037523	22500037523
cache.c1.xlarge	6501171200	650117120	650117120
cache.r3.large	14470348800	1468006400	1468006400
cache.r3.xlarge	30513561600	3040870400	3040870400
cache.r3.2xlarge	62495129600	6081740800	6081740800
cache.r3.4xlarge	126458265600	12268339200	12268339200
cache.r3.8xlarge	254384537600	24536678400	24536678400
cache.r4.large	13201781556	1320178155	1320178155
cache.r4.xlarge	26898228839	2689822883	2689822883
cache.r4.2xlarge	54197537997	5419753799	5419753799
cache.r4.4xlarge	108858546586	10885854658	10885854658

Node type	Maxmemory	Client-output-buffer-limit-slave-hard-limit	Client-output-buffer-limit-slave-soft-limit
cache.r4.8xlarge	218255432090	21825543209	21825543209
cache.r4.16xlarge	437021573120	43702157312	43702157312
cache.r5.large	14037181030	1403718103	1403718103
cache.r5.xlarge	28261849702	2826184970	2826184970
cache.r5.2xlarge	56711183565	5671118356	5671118356
cache.r5.4xlarge	113609865216	11360986522	11360986522
cache.r5.12xlarge	341206346547	34120634655	34120634655
cache.r5.24xlarge	682485973811	68248597381	68248597381
cache.r6g.large	14037181030	1403718103	1403718103
cache.r6g.xlarge	28261849702	2826184970	2826184970
cache.r6g.2xlarge	56711183565	5671118356	5671118356
cache.r6g.4xlarge	113609865216	11360986522	11360986522
cache.r6g.8xlarge	225000375228	22500037523	22500037523
cache.r6g.12xlarge	341206346547	34120634655	34120634655
cache.r6g.16xlarge	450000750456	45000075046	45000075046
cache.r6gd.xlarge	28261849702	2826184970	2826184970
cache.r6gd.2xlarge	56711183565	5671118356	5671118356
cache.r6gd.4xlarge	113609865216	11360986522	11360986522
cache.r6gd.8xlarge	225000375228	22500037523	22500037523

Node type	Maxmemory	Client-output-buffer-limit-slave-hard-limit	Client-output-buffer-limit-slave-soft-limit
cache.r6gd.12xlarge	341206346547	34120634655	34120634655
cache.r6gd.16xlarge	450000750456	45000075046	45000075046
cache.r7g.large	14037181030	1403718103	1403718103
cache.r7g.xlarge	28261849702	2826184970	2826184970
cache.r7g.2xlarge	56711183565	5671118356	5671118356
cache.r7g.4xlarge	113609865216	11360986522	11360986522
cache.r7g.8xlarge	225000375228	22500037523	22500037523
cache.r7g.12xlarge	341206346547	34120634655	34120634655
cache.r7g.16xlarge	450000750456	45000075046	45000075046
cache.m7g.large	6854542746	685454275	685454275
cache.m7g.xlarge	13891921715	1389192172	1389192172
cache.m7g.2xlarge	27966669210	2796666921	2796666921
cache.m7g.4xlarge	56116178125	5611617812	5611617812
cache.m7g.8xlarge	111325552312	11132555231	11132555231
cache.m7g.12xlarge	168715971994	16871597199	16871597199
cache.m7g.16xlarge	225000375228	22500037523	22500037523
cache.c7gn.large	3317862236	1403718103	1403718103
cache.c7gn.xlarge	6854542746	2826184970	2826184970
cache.c7gn.2xlarge	13891921715	5671118356	5671118356

Node type	Maxmemory	Client-output-buffer-limit-slave-hard-limit	Client-output-buffer-limit-slave-soft-limit
cache.c7gn.4xlarge	27966669210	11360986522	11360986522
cache.c7gn.8xlarge	56116178125	22500037523	22500037523
cache.c7gn.12xlarge	84357985997	34120634655	34120634655
cache.c7gn.16xlarge	113609865216	45000075046	45000075046

Note

All current generation instance types are created in an Amazon Virtual Private Cloud VPC by default.

T1 instances do not support Multi-AZ.

T1 and T2 instances do not support Redis OSS AOF.

Redis OSS configuration variables `appendonly` and `appendfsync` are not supported on Redis OSS version 2.8.22 and later.

Scaling ElastiCache (Redis OSS)

Scaling ElastiCache Serverless

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, 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 Redis OSS 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 can double ECPUs/second every 10-12 minutes. This scaling speed is sufficient for most workloads. 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 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 running 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 ElastiCache (Redis OSS) self-designed 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.

The following helps you find the correct topic for the scaling actions that you want to perform.

Scaling Redis OSS clusters

Action	Redis OSS (cluster mode disabled)	Redis OSS (cluster mode enabled)
Scaling in	Removing nodes from a cluster	Scaling clusters in Redis OSS (Cluster Mode Enabled)
Scaling out	Adding nodes to a cluster	Online resharding and shard rebalancing for Redis OSS (cluster mode enabled)
Changing node types	To a larger node type: • Scaling up single-node clusters for Redis OSS (Cluster Mode Disabled) • Scaling up Redis OSS clusters with replicas To a smaller node type: • Scaling down single-node Redis OSS clusters • Scaling down Redis OSS clusters with replicas	Online vertical scaling by modifying node type
Changing the number of node groups	Not supported for Redis OSS (cluster mode disabled) clusters	Scaling clusters in Redis OSS (Cluster Mode Enabled)

Topics

- Scaling clusters in Redis OSS (Cluster Mode Enabled)

Scaling clusters for Redis OSS (Cluster Mode Disabled)

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.

Contents

Scaling single-node clusters for Redis OSS (Cluster Mode Disabled)

Redis OSS (cluster mode disabled) nodes must be large enough to contain all the cache's data plus Redis OSS overhead. To change the data capacity of your 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 (Redis OSS) scaling up process is designed to make a best effort to retain your existing data and requires successful Redis OSS replication. For Redis OSS (cluster mode disabled) clusters, we recommend that sufficient memory be made available to Redis OSS.

You cannot partition your data across multiple Redis OSS (cluster mode disabled) clusters. However, if you only need to increase or decrease your cluster's read capacity, you can create a

Redis OSS (cluster mode disabled) cluster with replica nodes and add or remove read replicas. To create a Redis OSS (cluster mode disabled) cluster with replica nodes using your single-node Redis OSS cache cluster as the primary cluster, see Creating a Redis OSS (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, 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 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.

Topics

- Scaling up single-node clusters for Redis OSS (Cluster Mode Disabled)
- Scaling down single-node Redis OSS clusters

Scaling up single-node clusters for Redis OSS (Cluster Mode Disabled)

When you scale up a single-node Redis OSS cluster, ElastiCache performs the following process, whether you use the ElastiCache console, the AWS CLI, or the ElastiCache API.

1. A new cache cluster with the new node type is spun up in the same Availability Zone as the existing cache cluster.
2. The cache data in the existing cache cluster is copied to the new cache cluster. How long this process takes depends upon your node type and how much data is in the cache cluster.
3. Reads and writes are now served using the new cache cluster. Because the new cache cluster's endpoints are the same as they were for the old cache 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 cache 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 the r6gd node type, you can only scale to node sizes within the r6gd node family.

As shown in the following table, your 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 maintenance.

Blocked 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 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 Redis OSS scale up operation.
- Add the Redis OSS engine upgrade to this cache 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 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 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](#).

Scaling up single-node clusters for Redis OSS (Cluster Mode Disabled) (Console)

The following procedure describes how to scale up a single-node Redis OSS cluster using the ElastiCache Management Console. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale up a single-node 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 **Redis OSS clusters**.

3. From the list of clusters, choose the cluster you want to scale up (it must be running the Redis OSS engine, not the Clustered 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 Redis OSS cache clusters (AWS CLI)

The following procedure describes how to scale up a single-node Redis OSS cache cluster using the AWS CLI. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale up a single-node Redis OSS cache 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.m3.2xlarge",
         "cache.m3.large",
         "cache.m3.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r3.2xlarge",
         "cache.r3.4xlarge",
         "cache.r3.8xlarge",
         "cache.r3.large",
         "cache.r3.xlarge"
     ]
        "ScaleDownModifications": [
         "cache.t2.micro",
         "cache.t2.small ",
         "cache.t2.medium ",
            "cache.t1.small ",
     ],

 }
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Modify your existing cache cluster specifying the cache 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 cache cluster you are scaling up.
   - `--cache-node-type` – The new node type you want to scale the cache 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.m3.xlarge \
     --cache-parameter-group-name redis32-m2-xl \
     --apply-immediately
```

For Windows:

```
aws elasticache modify-cache-cluster ^
     --cache-cluster-id my-redis-cache-cluster ^
     --cache-node-type cache.m3.xlarge ^
     --cache-parameter-group-name redis32-m2-xl ^
     --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.m3.2xlarge"
            },
            "PreferredMaintenanceWindow": "tue:11:30-tue:12:30",
            "CacheNodeType": "cache.m3.medium",
             "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 cache 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 cache cluster.

   • `--cache-cache cluster-id` – The name of your single-node Redis OSS cache cluster. Use this parameter to describe a particular cache cluster rather than all cache 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 Redis OSS cache clusters (ElastiCache API)

The following procedure describes how to scale up a single-node Redis OSS cache cluster using the ElastiCache API. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale up a single-node Redis OSS cache 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 Redis OSS cache 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 cache cluster specifying the cache 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 cache cluster you are scaling up.
   - `CacheNodeType` – The new, larger node type you want to scale the cache cluster up to. This value must be one of the node types returned by the `ListAllowedNodeTypeModifications` action in step 1.
   - `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.m3.xlarge
    &CacheParameterGroupName redis32-m2-xl
```

```
    &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 cache cluster using the ElastiCache API `DescribeCacheClusters` action with the following parameter. When the status changes to *available*, you can begin using the new, larger cache cluster.

   - `CacheClusterId` – The name of your single-node Redis OSS cache cluster. Use this parameter to describe a particular cache cluster rather than all cache 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 Redis OSS clusters

The following sections walk you through how to scale a single-node 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 Redis OSS overhead is important to the long-term success of your new Redis OSS cluster. For more information, see Ensuring that you have enough memory to create a Redis OSS snapshot.

> **Note**
>
> For clusters running the r6gd node type, you can only scale to node sizes within the r6gd node family.

Topics

- Scaling down a single-node Redis OSS cluster (Console)
- Scaling down single-node Redis OSS cache clusters (AWS CLI)
- Scaling down single-node Redis OSS cache clusters (ElastiCache API)

Scaling down a single-node Redis OSS cluster (Console)

The following procedure walks you through scaling your single-node 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 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.

To scale down your single-node 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 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.
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 Redis OSS engine and not the Clustered 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 Redis OSS cache clusters (AWS CLI)

The following procedure describes how to scale down a single-node Redis OSS cache cluster using the AWS CLI.

To scale down a single-node Redis OSS cache 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.m3.2xlarge",
         "cache.m3.large",
         "cache.m3.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r3.2xlarge",
         "cache.r3.4xlarge",
         "cache.r3.8xlarge",
         "cache.r3.large",
         "cache.r3.xlarge"
     ]
        "ScaleDownModifications": [
         "cache.t2.micro",
         "cache.t2.small ",
         "cache.t2.medium ",
            "cache.t1.small ",
     ],

 }
```

For more information, see list-allowed-node-type-modifications in the *AWS CLI Reference*.

2. Modify your existing cache cluster specifying the cache 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 cache cluster you are scaling down.
   - `--cache-node-type` – The new node type you want to scale the cache 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.m3.xlarge \
        --cache-parameter-group-name redis32-m2-xl \
        --apply-immediately
   ```

   For Windows:

   ```
   aws elasticache modify-cache-cluster ^
        --cache-cluster-id my-redis-cache-cluster ^
        --cache-node-type cache.m3.xlarge ^
        --cache-parameter-group-name redis32-m2-xl ^
        --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.m3.2xlarge"
            },
            "PreferredMaintenanceWindow": "tue:11:30-tue:12:30",
            "CacheNodeType": "cache.m3.medium",
             "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 cache 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 cache cluster.

   - `--cache-cache cluster-id` – The name of your single-node Redis OSS cache cluster. Use this parameter to describe a particular cache cluster rather than all cache 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 Redis OSS cache clusters (ElastiCache API)

The following procedure describes how to scale updown a single-node Redis OSS cache cluster using the ElastiCache API.

To scale down a single-node Redis OSS cache 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 Redis OSS cache 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 cache cluster specifying the cache 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 cache cluster you are scaling down.
   - `CacheNodeType` – The new, smaller node type you want to scale the cache cluster down to. This value must be one of the node types returned by the `ListAllowedNodeTypeModifications` action in step 1.
   - `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.m3.xlarge
    &CacheParameterGroupName redis32-m2-xl
    &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 cache cluster using the ElastiCache API `DescribeCacheClusters` action with the following parameter. When the status changes to *available*, you can begin using the new, smaller cache cluster.

- `CacheClusterId` – The name of your single-node Redis OSS cache cluster. Use this parameter to describe a particular cache cluster rather than all cache 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 Redis OSS (Cluster Mode Disabled) clusters with replica nodes

A 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 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 Redis OSS (cluster mode disabled) does not support partitioning your data across multiple clusters, each node in a Redis OSS (cluster mode disabled) replication group contains the entire cache dataset. 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 Redis OSS replication. For Redis OSS clusters with replicas, we recommend that sufficient memory be made available to Redis OSS.

Related Topics

- High availability using replication groups
- Replication: Redis OSS (Cluster Mode Disabled) vs. Redis OSS (Cluster Mode Enabled)
- Minimizing downtime in ElastiCache (Redis OSS) with Multi-AZ
- Ensuring that you have enough memory to create a Redis OSS snapshot

Topics

- Scaling up Redis OSS clusters with replicas
- Scaling down Redis OSS clusters with replicas
- Increasing read capacity
- Decreasing read capacity

Scaling up Redis OSS clusters with replicas

Amazon ElastiCache provides console, CLI, and API support for scaling your 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 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.
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 Redis OSS scale-up operation is blocked if you have an engine upgrade scheduled for the cluster’s next maintenance window.

Blocked 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 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 Redis OSS scale-up operation.
- Add the Redis OSS engine upgrade to this cache 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 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 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.

Scaling up a 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 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 **Redis OSS clusters**
3. From the list of clusters, choose the cluster you want to scale up. This cluster must be running the Redis OSS engine and not the Clustered 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 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 (Redis OSS) 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 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 cache cluster.

To scale up a 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.m3.2xlarge",
         "cache.m3.large",
         "cache.m3.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r3.2xlarge",
         "cache.r3.4xlarge",
         "cache.r3.8xlarge",
         "cache.r3.large",
         "cache.r3.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 cache clusters in this replication group. This value must be one of the instance 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 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.m3.xlarge \
     --cache-parameter-group-name redis32-m3-2xl \
     --apply-immediately
```

   For Windows:

```
aws elasticache modify-replication-group ^
     --replication-group-id my-repl-group ^
     --cache-node-type cache.m3.xlarge ^
     --cache-parameter-group-name redis32-m3-2xl \
     --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 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 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 cache cluster.

To scale up a 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 `ModifyRedplicationGroup` ElastiCache API action and with the following parameters.

   - `ReplicationGroupId` – the name of the replication group.
   - `CacheNodeType` – the new, larger node type of the cache clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in step 1.
   - `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.m3.2xlarge
    &CacheParameterGroupName=redis32-m3-2xl
    &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 Redis OSS clusters with replicas

The following sections walk you through how to scale a Redis OSS (cluster mode disabled) cache 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 that you have enough memory to create a Redis OSS snapshot.

> **Note**
>
> For clusters running the r6gd node type, you can only scale to node sizes within the r6gd node family.

> **Important**
>
> If your parameter group uses `reserved-memory` to set aside memory for 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.

Scaling down a Redis OSS cluster with replicas (Console)

The following process scales your Redis OSS cluster with replica nodes to a smaller node type using the ElastiCache console.

To scale down a 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 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.

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 Redis OSS engine and not the Clustered 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 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 (Redis OSS) 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 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 cache 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 cache cluster.

To scale down a 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.m3.2xlarge",
         "cache.m3.large",
         "cache.m3.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r3.2xlarge",
         "cache.r3.4xlarge",
         "cache.r3.8xlarge",
         "cache.r3.large",
         "cache.r3.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 cache clusters in this replication group. This value must be one of the instance 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 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 redis32-m3-2xl \
     --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 redis32-m3-2xl \
     --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 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 (Redis OSS) 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 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 cache 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 cache cluster.

To scale down a 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 `ModifyRedplicationGroup` ElastiCache API action and with the following parameters.

   - `ReplicationGroupId` – the name of the replication group.
   - `CacheNodeType` – the new, smaller node type of the cache clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in step 1.
   - `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.m3.2xlarge
    &CacheParameterGroupName=redis32-m3-2xl
    &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 Redis OSS replication group.

You can scale your 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 Redis OSS (Cluster Mode Disabled) replication groups.

Decreasing read capacity

To decrease read capacity, delete one or more read replicas from your 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 Redis OSS (Cluster Mode Disabled) replication groups .

Scaling clusters in Redis OSS (Cluster Mode Enabled)

As demand on your clusters changes, you might decide to improve performance or reduce costs by changing the number of shards in your 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 *BytesUseForCache*.
- **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*, *BytesUseForCache*, *CPUUtilization*, *NetworkBytesIn*, *NetworkBytesOut*, *CurrConnections*, and *NewConnections*.

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 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 Redis OSS overhead.

For more information, see Select cache node size.

Contents

Offline resharding and shard rebalancing for 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 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 Redis OSS clusters with data tiering enabled. For more information, see Data tiering.

- 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 Redis OSS (cluster mode enabled) cluster offline

1. Create a manual backup of your existing 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.

Online resharding and shard rebalancing for Redis OSS (cluster mode enabled)

By using online resharding and shard rebalancing with Amazon ElastiCache (Redis OSS) version 3.2.10 or newer, you can scale your ElastiCache for Redis OSS (cluster mode enabled) 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 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 Redis OSS (cluster mode enabled) cluster.
- **Rebalance –** Move the keyspaces among the shards in your ElastiCache for 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 (Redis OSS) online resharding and rebalancing:

- These processes require Redis OSS engine version 3.2.10 or newer. For information on upgrading your engine version, see Engine versions and upgrading .
- 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 (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 Redis OSS Functions, we recommend loading all of your functions to all the shards before scaling out so that your ElastiCache for Redis OSS 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 ElastiCache for 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 Redis OSS (cluster mode enabled) cluster using the AWS Management Console, AWS CLI, or ElastiCache API. When you add shards to a 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 Redis OSS (cluster mode enabled) cluster. The following procedure describes the process.

To add shards to your Redis OSS (cluster mode enabled) cluster

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Redis OSS clusters**.
3. Locate and choose the name, not the box to the left of the cluster's name, of the Redis OSS (cluster mode enabled) cluster that you want to add shards to.

> **Tip**
>
> Redis OSS (cluster mode enabled) show **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 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 the Redis OSS (cluster mode enabled) cluster `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 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 Redis OSS (cluster mode enabled) cluster by adding shards using the ElastiCache API.

Example - Adding Shards

The following example adds node groups to the 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 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 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 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 Redis OSS (cluster mode enabled) cluster

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Redis OSS clusters**.
3. Locate and choose the name, not the box to the left of the cluster's name, of the Redis OSS (cluster mode enabled) cluster you want to remove shards from.

   > **Tip**
   >
   > 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 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 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 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 Redis OSS (cluster mode enabled) cluster online by using the `ModifyReplicationGroupShardConfiguration` operation.

The following process describes how to reconfigure the shards in your 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 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 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 Redis OSS (cluster mode enabled) cluster using the AWS Management Console, AWS CLI, or ElastiCache API.

Topics

Online Shard Rebalancing (Console)

The following process describes how to reconfigure the shards in your Redis OSS (cluster mode enabled) cluster by rebalancing shards using the AWS Management Console.

To rebalance the keyspaces among the shards on your Redis OSS (cluster mode enabled) cluster

1. Open the ElastiCache console at https://console.aws.amazon.com/elasticache/.
2. From the navigation pane, choose **Redis OSS clusters**.
3. Choose the name, not the box to the left of the name, of the Redis OSS (cluster mode enabled) cluster that you want to rebalance.

   > **Tip**
   >
   > 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 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 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 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 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 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 Amazon ElastiCache (Redis OSS) version 3.2.10 or newer, you can scale your Redis OSS clusters dynamically with minimal downtime. This allows your 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.

You can do the following:

- **Scale up** – Increase read and write capacity by adjusting the node type of your 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.
> - 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

Topics

Scaling up Redis OSS cache clusters (Console)

The following procedure describes how to scale up a Redis OSS cluster using the ElastiCache Management Console. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale up a 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 **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 Redis OSS cache clusters (AWS CLI)

The following procedure describes how to scale up a Redis OSS cache cluster using the AWS CLI. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale up a Redis OSS cache 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.m3.2xlarge",
         "cache.m3.large",
         "cache.m3.xlarge",
         "cache.m4.10xlarge",
         "cache.m4.2xlarge",
         "cache.m4.4xlarge",
         "cache.m4.large",
         "cache.m4.xlarge",
         "cache.r3.2xlarge",
         "cache.r3.4xlarge",
         "cache.r3.8xlarge",
         "cache.r3.large",
         "cache.r3.xlarge"
     ]
        "ScaleDownModifications": [
         "cache.t2.micro",
         "cache.t2.small ",
         "cache.t2.medium",
         "cache.t1.small "
     ],
}
```

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 cache 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.m3.xlarge \
        --apply-immediately
   ```

   For Windows:

   ```
   aws elasticache modify-replication-group ^
        --replication-group-id my-redis-cluster ^
        --cache-node-type cache.m3.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.m3.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 cache 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 cache cluster node.

Scaling up Redis OSS cache clusters (ElastiCache API)

The following process scales your cache cluster from its current node type to a new, larger node type using the ElastiCache API. During this process, ElastiCache (Redis OSS) 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 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 cache cluster.

To scale up a 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 cache clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in step 1.
- `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.m3.2xlarge
    &CacheParameterGroupName=redis32-m3-2xl
    &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 Redis OSS cache clusters (Console)
- Scaling down Redis OSS cache clusters (AWS CLI)
- Scaling down Redis OSS cache clusters (ElastiCache API)

Scaling down Redis OSS cache clusters (Console)

The following procedure describes how to scale down a Redis OSS cluster using the ElastiCache Management Console. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale Down a 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 **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 Redis OSS cache clusters (AWS CLI)

The following procedure describes how to scale down a Redis OSS cache cluster using the AWS CLI. During this process, your Redis OSS cluster will continue to serve requests with minimal downtime.

To scale down a Redis OSS cache 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.m3.2xlarge",
            "cache.m3.large",
            "cache.m3.xlarge",
            "cache.m4.10xlarge",
            "cache.m4.2xlarge",
            "cache.m4.4xlarge",
            "cache.m4.large",
   ```

```
            "cache.m4.xlarge",
            "cache.r3.2xlarge",
            "cache.r3.4xlarge",
            "cache.r3.8xlarge",
            "cache.r3.large",
            "cache.r3.xlarge"
        ]

           "ScaleDownModifications": [
            "cache.t2.micro",
            "cache.t2.small ",
            "cache.t2.medium ",
            "cache.t1.small"
        ]
}
```

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 cache 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 cache 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 cache cluster node.

Scaling down Redis OSS cache 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 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 cache 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 cache clusters in this replication group. This value must be one of the instance types returned by the `ListAllowedNodeTypeModifications` action in step 1.
   - `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=redis32-m3-2xl
    &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 JSON in ElastiCache (Redis OSS)

ElastiCache (Redis OSS) supports the native JavaScript Object Notation (JSON) format, which is a simple, schemaless way to encode complex datasets inside Redis OSS clusters. You can natively store and access data using the JavaScript 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.

In addition to using 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. This is API compatible and RDB compatible with version 2 of the RedisJSON module, so you can easily migrate existing JSON-based Redis OSS applications into ElastiCache (Redis OSS). For more information on the supported Redis OSS commands, see Supported Redis OSS JSON 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 Redis OSS.

> **Note**
>
> To use JSON, you must be running Redis OSS engine version 6.2.6 or later.

Topics

- Redis OSS JSON data type overview
- Supported Redis OSS JSON commands

Redis OSS JSON data type overview

ElastiCache (Redis OSS) supports a number of 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 Redis OSS commands that are supported.

Terminology

Term	Description
JSON document	Refers to the value of a Redis OSS 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 Redis OSS commands are members of the @json category. All JSON commands enforce any keyspace or command restrictions and permissions.

- There are five existing 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 Redis OSS 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 Redis OSS.

To query core metrics, run the following Redis OSS command:

```
info json_core_metrics
```

How ElastiCache (Redis OSS) interacts with JSON

The following section describes how ElastiCache (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 (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 Redis OSS JSON commands.

Direct array filtering

ElastiCache (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 (Redis OSS) would return [1,2,3] in both circumstances.

Array indexing behavior

ElastiCache (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 (Redis OSS) 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

MemoryDB 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 Redis OSS JSON commands

ElastiCache (Redis OSS) supports the following Redis OSS JSON commands:

Topics

JSON.ARRAPPEND

Appends one or more values to the array values at the path.

Syntax

```
JSON.ARRAPPEND <key> <path> <json> [json ...]
```

- key (required) – A 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.
- `SYNTAXERR` error if one of the input json arguments is not a valid JSON string.
- `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 last 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 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 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 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:

- Array of bulk strings. Each element is a key name in the object.

- 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.
- `WRONGTYPE` 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\"]]'
(error) SYNTAXERR Failed to parse JSON string due to syntax error
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 $[3]
1) (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 [*]
(integer) 0
127.0.0.1:6379> JSON.ARRLEN k2 $[1]
1) (nil)
127.0.0.1:6379> JSON.ARRLEN k2 $[2]
1) (integer) 2
```

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 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.ARRINSERT <key> <path> <start> <end>
```

- key (required) – A 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 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 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 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 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 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 Redis OSS.

Syntax

```
JSON.DEL <key> [path]
```

- key (required) – A 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 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 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 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.NUMINCRBY

Increments the number values at the path by a given number.

Syntax

```
JSON.NUMINCRBY <key> <path> <number>
```

- key (required) – A 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 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 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.
- `WRONGTYPE` 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 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.
- `WRONGTYPE` 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.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 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 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 Redis OSS key of JSON document type.
- path (required) – A JSON path. For a new Redis OSS key, the JSON path must be the root ".".
- NX (optional) – If the path is the root, set the value only if the Redis OSS 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 Redis OSS 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 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 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 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 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.

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 cache clusters in the

replication group. If the primary cache 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 cache cluster tags will be added to the replication group and propagated to all cache 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 cache clusters.

- **CreateCacheCluster** :
  - If the `--replication-group-id` is supplied:

    If tags are included in the request, the cache cluster will be tagged only with those tags. If no tags are included in the request, the cache cluster will inherit the replication group tags instead of the primary cache cluster's tags.
  - If the `--snapshot-name` is supplied:

    If tags are included in the request, the cache cluster will be tagged only with those tags. If no tags are included in the request, the snapshot tags will be added to the cache 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 cache 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

```
aws elasticache create-serverless-cache \
    --serverless-cache-name CacheName \
    --engine redis
    --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 redis \
--tags Key="project",Value="XYZ" Key="Elasticache",Value="Service"
```

- Creating a serverless snapshot with tags.

```
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.

  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.

```
{
    "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.

```
{
    "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.

```
{
    "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`.

```
{
    "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.

```
{
    "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`.

```
{
    "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 Redis OSS nodes. When you add, list, modify, copy, or remove a tag, the operation is applied only to the specified node.

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.

Sample arn: `arn:aws:elasticache:us-west-2:1234567890:cluster:my-cluster`

Cost allocation tags are applied to ElastiCache (Redis OSS) nodes. The node 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

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 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.

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 a node in an ElastiCache (Redis OSS) 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 node in an ElastiCache (Redis OSS) cluster by using the remove-tags-from-resource operation.

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

You can use the ElastiCache API to list tags on an existing resource by using the ListTagsForResource operation.

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 to the resource `my-cluster-001` in the us-west-2 region.

```
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 (Redis OSS) node 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.

Topics

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?

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 (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 Amazon ElastiCache for Redis OSS (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 Redis OSS client library by connecting to the ElastiCache reader endpoint for cluster mode disabled, or by using the Redis OSS READONLY command for cluster mode enabled.

  [Resources]:
  - Finding connection endpoints
  - Cluster Right-Sizing
  - Redis OSS 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 for Redis OSS 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 Amazon ElastiCache for Redis OSS using Amazon CloudWatch
  - Scaling ElastiCache for Redis OSS Clusters
  - Scaling ElastiCache for Memcached Clusters
- **[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 for Redis OSS with CloudWatch Metrics
  - Monitoring Memcached with CloudWatch Metrics
  - Monitoring best practices with Amazon ElastiCache for Redis OSS 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 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 for Redis OSS client library using the ElastiCache reader endpoint for cluster mode disabled. For cluster mode enabled, use the ElastiCache for Redis OSS READONLY command. For many of the ElastiCache for Redis OSS client libraries, ElastiCache for Redis OSS READONLY is implemented by default or via a configuration setting.

  [Resources]:

  - Finding connection endpoints
  - 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 for Redis OSS 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 Amazon ElastiCache for Redis OSS 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 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. Amazon ElastiCache for Redis OSS can emit Redis OSS engine logs to both Amazon CloudWatch and Amazon Kinesis Data Firehose.

Question-level benefit: Enabling Redis OSS engine logs on Amazon ElastiCache for Redis OSS clusters provides insight into events that impact the health and performance of clusters. Redis OSS engine logs provide data directly from the Redis OSS engine that is not available through the ElastiCache events mechanism. Through careful observation of both ElastiCache events (see preceding OE-1) and Redis OSS engine logs, it is possible to determine an order of events when troubleshooting from both the ElastiCache service perspective and Redis OSS engine perspective.

- **[Required]** Ensure that Redis OSS engine logging functionality is enabled, which is available as of ElastiCache for Redis OSS 6.2 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 Redis OSS engine log for important information.

  As an example, create a query against the CloudWatch Log group that contains the 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?

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 (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 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 ElastiCache (Redis OSS) clusters at an individual client level, it is recommended to authenticate via the ElastiCache (Redis OSS) AUTH command. ElastiCache (Redis OSS) authentication tokens, with optional user and user group management, enable ElastiCache (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) 6.x and higher, 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 Redis OSS client that supports TLS to be able to leverage RBAC.

  [Resources]:

  - Applying RBAC to a Replication Group for ElastiCache (Redis OSS)
  - Specifying Permissions Using an Access String
  - ACL
  - Supported ElastiCache (Redis OSS) versions
- **[Best]** For ElastiCache (Redis OSS) versions prior to 6.x, in addition to setting strong token/password and maintaining a strict password policy for ElastiCache (Redis OSS) 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 (Redis OSS) 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 Redis OSS commands that can have adverse impacts on operations if executed by mistake or by malicious actors. These commands can have unintended 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 (Redis OSS) 5.0.3, 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 (Redis OSS) version 5.0.3 (deprecated, use version 5.0.6)

- Redis OSS 5.0.3 parameter changes
- Redis OSS security

SEC 4: How do you ensure data encryption at rest with ElastiCache

Question-level introduction: While ElastiCache (Redis OSS) 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 (Redis OSS) 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 Conditions
  - 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 (Redis OSS) 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 (Redis OSS) 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 an ElastiCache (Redis OSS) 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 (Redis OSS), allowing for tighter control to manage the creation, modification and deletion of ElastiCache (Redis OSS) 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 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 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 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

Topics

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 (Memcached) vs ElastiCache (Redis OSS)

  1. ElastiCache (Memcached) does not provide any replication mechanism and is used primarily for ephemeral workloads.
  2. ElastiCache (Redis OSS) offers HA features discussed below
- **[Best]** For workloads that require HA, use ElastiCache (Redis OSS) 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 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).

   Amazon ElastiCache for Redis OSS 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 ElastiCache (Redis OSS) 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 Amazon 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 (Redis OSS) 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 (Redis OSS)
  - Making manual backups
  - Scheduling automatic backups
  - Backup and Restore ElastiCache (Redis OSS) 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 (Redis OSS) offers several options to implement disaster recovery based on workload resilience requirements. With Amazon ElastiCache Global Datastore, you can write to your ElastiCache (Redis OSS) 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 (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 (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 (Redis OSS) 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 (Redis OSS) 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 (Redis OSS) with Multi-AZ
  - Review the best practices of your ElastiCache (Redis OSS) 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 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 Redis OSS WAIT command to force replicas to acknowledge writes before responding to connected clients.

  [Resources]:

  - Metrics for Redis OSS
  - Monitoring best practices with Amazon ElastiCache for Redis OSS 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 Amazon ElastiCache (Redis OSS)
  - 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 (Redis OSS) clusters
  - Online scaling up
  - Scaling ElastiCache for Memcached clusters

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?

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 (Redis OSS) workloads, rename computationally expensive commands, such as KEYS, to limit the ability of users to run blocking commands on production clusters.
  - ElastiCache (Redis OSS) workloads running engine 6.x, 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 an ElastiCache for Redis OSS 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 Amazon ElastiCache for Redis OSS cluster parameter group.
- **[Better]** Analyze slow queries and look for optimization techniques.

  - For ElastiCache (Redis OSS) workloads, learn more about your queries by analyzing the Slow Log. For example, you can use the following command, `redis-cli slowlog get 10` to show last 10 commands which exceeded latency thresholds (10 seconds by default).
  - Certain queries can be performed more efficiently using complex ElastiCache (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 (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
  - Monitoring use with CloudWatch Metrics
  - Using Amazon CloudWatch alarms
  - Redis-specific parameters
  - SLOWLOG
  - Redis OSS 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.
  - Amazon ElastiCache for 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 (Redis OSS) cluster mode enabled to improve scalability.
  - ElastiCache (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 (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 Redis OSS data structures such a lists, streams, sets, etc. These data structures are stored in single Redis OSS 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 (Redis OSS) workloads, you can detect hot keys using `redis-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 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.
  - EElastiCache (Redis OSS) version 6 supports 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 (Redis OSS) for higher availability
  - Finding connection endpoints
  - Load balancing best practices

- Online resharding and shard rebalancing for Redis OSS (cluster mode enabled)
- Client-side caching in 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 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 ElastiCache (Redis OSS) INFO command and do not include the network and I/O time. This is only the time consumed by ElastiCache (Redis OSS) to process the operations.

  [Resources]:

  - Metrics for Redis OSS
  - Monitoring best practices with Amazon ElastiCache for Redis OSS 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
- 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 (Redis OSS) and ElastiCache (Memcached) 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]:**
  - [Amazon ElastiCache for Redis OSS - Global Datastore]
  - [Memcached specific parameters]
  - [Amazon ElastiCache for Redis OSS 5.0.3 enhances I/O handling to boost performance]
  - [Metrics for Redis OSS]
  - [Configure Amazon ElastiCache for Redis OSS 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. Amazon ElastiCache for Redis OSS 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 ElastiCache (Redis OSS) 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 (Redis OSS) workloads, when explicitly removing keys from the cluster, `UNLINK` is like `DEL`: it removes the specified keys. However, the command performs the actual memory

reclaiming in a different thread, so it is not blocking, while `DEL` is. The actual removal will happen later asynchronously.

- For ElastiCache (Redis OSS) 6.x 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 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 ElastiCache (Redis OSS) 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 ElastiCache (Redis OSS) 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 (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 (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. Amazon ElastiCache for Redis OSS 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]** Amazon ElastiCache (Redis OSS) running 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 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 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 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 redis-cli via the SLOWLOG command, consider increasing this value. This allows more commands to remain in 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 Redis OSS memory.
- **[Resources]:**
  - How do I turn on Redis OSS Slow log in an ElastiCache (Redis OSS) cache 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 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 (Redis OSS) 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 (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 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

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?

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 Redis OSS clusters.
  2. Only limited instance types support data tiering.
  3. Only ElastiCache for Redis OSS version 6.2 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 ration 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

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. Learn more about connecting to a TLS enabled cache here.
2. **VPC:** ElastiCache caches are accessible only from within a VPC. 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.
3. **Security groups:** ElastiCache uses security groups to control access to your cache. 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 self-designed). ElastiCache uses these ports to accept Redis OSS commands. Learn more about how to setup port access here.

Redis OSS client errors

ElastiCache Serverless is only accessible using Redis OSS clients that support the Redis OSS cluster mode protocol. Self-designed clusters can be accessed from Redis OSS clients in either mode, depending on the cluster configuration.

If you are experiencing Redis OSS errors in your client, consider the following:

1. **Cluster mode:** If you are experiencing CROSSLOT errors or errors with the SELECT Redis OSS command, you may be trying to access a Cluster Mode Enabled cache with a Redis OSS client that does not support the Redis OSS Cluster protocol. ElastiCache Serverless only supports Redis OSS clients that support the Redis OSS cluster protocol. If you want to use Redis OSS in "Cluster Mode Disabled" (CMD), then you must design your own 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 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 AWS Health Dashboard and your Personal Health Dashboard for more information. If necessary, consider opening a support case with AWS 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 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.
- **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 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 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 `ThrottledRequests` ElastiCache Serverless metric. Review the section below for troubleshooting throttled requests.
- **Uniform distribution of keys and requests:** In ElastiCache (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 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 `ThrottledRequests` 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 (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, in a workload that executes simple SET/GET commands.

- **Read from Replica:** If you application allows it, consider using the “Read from Replica“ feature. Most 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.

Related Topics

- Additional troubleshooting steps
- the section called “Best practices and caching strategies”

Additional troubleshooting steps

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 Redis OSS. 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 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
- 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
- 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: 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: Redis OSS is a multi-threaded application. 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 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]() 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. Redis OSS provides 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 (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 https://redis.io/commands.

Examples of potential issues:

- Lua scripts: Redis OSS provides an embedded Lua interpreter, allowing the execution of scripts on the server-side. Lua scripts on 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 Redis OSS 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 (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 (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 https://redis.io/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 (Redis OSS) or 11211 for ElastiCache (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 shaped 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 shaped 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 (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 Redis OSS 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 (Redis OSS) versions earlier than 2.8.22 use a forked process to create backups and process full syncs with the replicas. This method may incur in significant memory overhead for write intensive use-cases.

  Starting with ElastiCache Redis OSS 2.8.22, AWS introduced a forkless backup and replication method. The new method may delay writes in order to prevent failures. Both methods can cause periods of higher CPU utilization, lead to higher response times and consequently lead to client timeouts during their 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 (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 Redis OSS 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 (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 (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 shaped 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.

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

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.

Topics

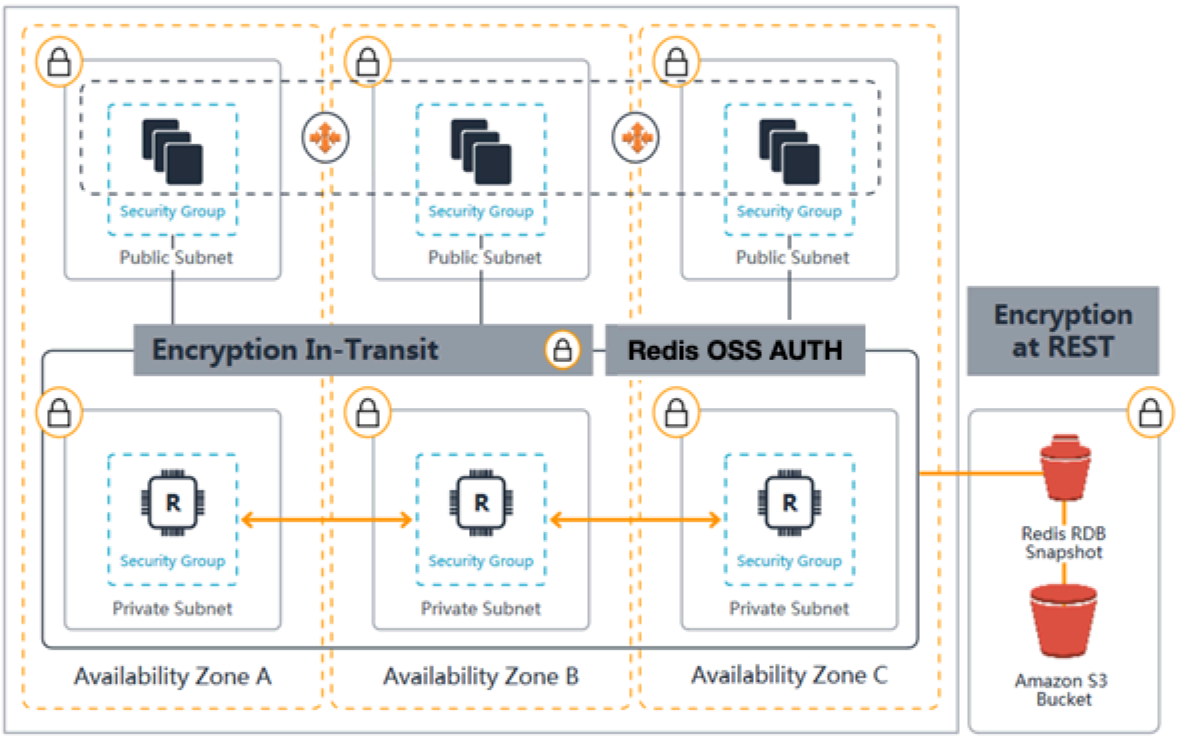
Data security in Amazon ElastiCache

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 (Redis OSS) provides encryption features for data on caches running Redis OSS versions 3.2.6 (scheduled for EOL, see Redis OSS versions end of life schedule), 4.0.10 or later:

- 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.

Amazon ElastiCache (Redis OSS) also supports authenticating users with either IAM or Redis OSS AUTH, and authorizing user operations using Role-Based Access Control (RBAC).



ElastiCache (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 serverless caches have in-transit encryption enabled. For self-designed 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.

Topics

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 Redis OSS AUTH feature, the server can authenticate the clients.

In-transit encryption conditions

The following constraints on Amazon ElastiCache in-transit encryption should be kept in mind when you plan your self-designed cluster implementation:

- In-transit encryption is supported on replication groups running Redis OSS versions 3.2.6, 4.0.10 and later.
- Modifying the in-transit encryption setting, for an existing cluster, is supported on replication groups running Redis OSS version 7 and later.
- In-transit encryption is supported only for replication groups running in an Amazon VPC.
- In-transit encryption is not supported for replication groups running the following node types: M1, M2.

  For more information, see Supported node types.
- 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.
- Usage of old TLS 1.0 and TLS 1.1 is deprecated across all AWS Regions for ElastiCache version 6 and above. ElastiCache will continue to support TLS 1.0 and 1.1 until May 8, 2025. Customers must update their client software before that date.

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.

See also

- At-Rest Encryption in ElastiCache
- Authenticating with the Redis OSS AUTH command
- Authenticating Users with Role-Based Access Control (RBAC)
- Amazon VPCs and ElastiCache security
- Identity and Access Management for Amazon ElastiCache

Enabling in-transit encryption

All serverless caches have in-transit encryption enabled. On a self-designed 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 self-designed 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:

- Choose engine version 3.2.6, 4.0.10 or later.
- Click the checkbox next to **Enable** for the **Encryption in transit** option.

For the step-by-step process, see the following:

- Creating a Redis OSS (cluster mode disabled) cluster (Console)
- Creating a Redis OSS (cluster mode enabled) cluster (Console)

Enabling in-transit encryption for an existing self-designed 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 Redis OSS clients to connect using both encrypted and unencrypted connections. After you migrate all your 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.

Step 1: 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 **Redis OSS caches** from the ElastiCache **Resources** listed on the navigation pane, present on the left hand.
3. Choose the **Redis OSS 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 Redis OSS clients to use encrypted connections:

Step 2: 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 **Redis OSS caches** from the ElastiCache **Resources** listed on the navigation pane, present on the left hand.
3. Choose the **Redis OSS 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 Redis OSS replication group using the AWS CLI, use the parameter `transit-encryption-enabled`.

Enabling in-transit encryption on a new self-designed cluster for Redis OSS (Cluster Mode Disabled) (CLI)

Use the AWS CLI operation `create-replication-group` and the following parameters to create a Redis OSS replication group with replicas that has in-transit encryption enabled:

Key parameters:

- **`--engine`**—Must be `redis`.
- **`--engine-version`**—Must be 3.2.6, 4.0.10 or later.
- **`--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 Redis OSS (Cluster Mode Disabled) replication group from scratch (AWS CLI)
- create-replication-group

Enabling in-transit encryption on a new self-designed cluster for Redis OSS (Cluster Mode Enabled) (CLI)

Use the AWS CLI operation `create-replication-group` and the following parameters to create a Redis OSS (cluster mode enabled) replication group that has in-transit encryption enabled:

Key parameters:

- **`--engine`**—Must be `redis`.
- **`--engine-version`**—Must be 3.2.6, 4.0.10 or later.
- **`--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 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 Redis OSS clients to connect using both encrypted and unencrypted connections. After you migrate all your 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 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 Amazon ElastiCache (Redis OSS) with in-transit encryption using redis-cli

To access data from ElastiCache (Redis OSS) caches enabled with in-transit encryption, you use clients that work with Secure Socket Layer (SSL). You can also use redis-cli with TLS/SSL on Amazon Linux and Amazon Linux 2. 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 redis-cli to connect to a Redis OSS cluster enabled with in-transit encryption on Amazon Linux 2023, Amazon Linux 2, or Amazon Linux, follow these steps.

1. Download and compile the redis-cli utility. This utility is included in the Redis OSS software distribution.
2. At the command prompt of your EC2 instance, type the appropriate commands for the version of Linux you are using.

   Amazon Linux 2023

   If using Amazon Linux 2023, enter this:

```
sudo yum install redis6 -y
```

Then type the following command, substituting the endpoint of your cluster and port for what is shown in this example.

```
redis-cli -h Primary or Configuration Endpoint --tls -p 6379
```

For more information on finding the endpoint, see Find your Node Endpoints.

Amazon Linux 2

If using Amazon Linux 2, enter this:

```
sudo yum -y install openssl-devel gcc
wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make distclean
make redis-cli BUILD_TLS=yes
sudo install -m 755 src/redis-cli /usr/local/bin/
```

Amazon Linux

If using Amazon Linux, enter this:

```
sudo yum install gcc jemalloc-devel openssl-devel tcl tcl-devel clang wget
wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make redis-cli CC=clang BUILD_TLS=yes
sudo install -m 755 src/redis-cli /usr/local/bin/
```

On Amazon Linux, you may also need to run the following additional steps:

```
sudo yum install clang
CC=clang make
sudo make install
```

3. After you have downloaded and installed the redis-cli utility, it is recommended that you run the optional `make-test` command.

4. To connect to a cluster with encryption and authentication enabled, enter this command:

```
redis-cli -h Primary or Configuration Endpoint --tls -a 'your-password' -p 6379
```

> **Note**
>
> If you install redis6 on Amazon Linux 2023, you can now use the command `redis6-cli` instead of `redis-cli`:
>
> ```
> redis6-cli -h Primary or Configuration Endpoint --tls -p 6379
> ```

Encrypted connection with stunnel

To use redis-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/redis-cli.conf'` simultaneously to add a ElastiCache (Redis OSS) cluster endpoint to one or more connection parameters, using the provided output below as template.

```
vi /etc/stunnel/redis-cli.conf


fips = no
setuid = root
setgid = root
pid = /var/run/stunnel.pid
debug = 7
delay = yes
options = NO_SSLv2
options = NO_SSLv3
[redis-cli]
```

```
    client = yes
    accept = 127.0.0.1:6379
    connect = primary.ssltest.wif01h.use1.cache.amazonaws.com:6379
[redis-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 `redis-cli` and the `redis-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.

3. Start `stunnel`.

```
sudo stunnel /etc/stunnel/redis-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 (Redis OSS) cluster creation, this example uses the redis-cli to connect to the ElastiCache (Redis OSS) server using complete path for redis-cli, on Amazon Linux:

```
/home/ec2-user/redis-stable/src/redis-cli -h localhost -p 6379
```

If AUTH password was used during Redis OSS cluster creation, this example uses redis-cli to connect to the Redis OSS server using complete path for redis-cli, on Amazon Linux:

```
/home/ec2-user/redis-stable/src/redis-cli -h localhost -p 6379 -a my-secret-password
```

OR

- Change directory to redis-stable and do the following:

  If no AUTH password was used during ElastiCache (Redis OSS) cluster creation, this example uses the redis-cli to connect to the ElastiCache (Redis OSS) server using complete path for redis-cli, on Amazon Linux:

  ```
  src/redis-cli -h localhost -p 6379
  ```

  If AUTH password was used during Redis OSS cluster creation, this example uses redis-cli to connect to the Redis OSS server using complete path for redis-cli, on Amazon Linux:

  ```
  src/redis-cli -h localhost -p 6379 -a my-secret-password
  ```

This example uses Telnet to connect to the 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 self-designed 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 Redis OSS Clusters with version 7.0 or above. So if you have a cluster running 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 version behavior and compatibility differences.

Topics

Define the string constants that will launch the ElastiCache 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 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 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 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 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 per-node DNS records are kept and the new per-node DNS records are being generated in a different format. TLS-enabled clusters use a different format of DNS records than non-TLS-enabled clusters. ElastiCache will keep both DNS records when a cluster is configured in encryption mode: Preferred so that Applications and other 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 endpoints for non-TLS enabled cluster will remain active. There will be no downtime when cluster is re-configured form TLS encryption mode 'none' to 'preferred'.
- New TLS 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 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 they might change when adding or removing shards.
- Don't hardcode 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 this 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 Redis OSS engine using the Redis OSS 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
```

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. When enabled, it encrypts the following aspects:

- Disk during sync, backup and swap operations
- Backups stored in Amazon S3

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.

Note

The default (service managed) encryption is the only option available in the GovCloud (US) Regions.

Important

Enabling at-Rest Encryption on an existing self-designed 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

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 Redis OSS versions (3.2.6 scheduled for EOL, see Redis OSS versions end of life schedule), 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.
  - R6gd, R6g, R5, R4, R3
  - M6g, M5, M4, M3
  - T4g,T3, T2

  For more information, see Supported node types
- At-rest encryption is enabled by explicitly setting the parameter `AtRestEncryptionEnabled` to `true`.
- 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 node type from the r6gd family, data stored on SSD is encrypted whether at-rest encryption is enabled or not.
- The option to use customer managed key for encryption at rest is not available in AWS GovCloud (us-gov-east-1 and us-gov-west-1) regions.
- If a cluster is using a node type from the r6gd family, data stored on SSD is encrypted with the chosen customer managed AWS KMS key (or service-managed encryption in AWS GovCloud Regions).

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.

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.
- 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 self-designed 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 on an Existing Self-Designed Redis OSS Cluster

You can only enable at-rest encryption when you create a 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 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 Self-Designed 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 3.2.6, 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 Redis OSS (cluster mode disabled) cluster (Console)
- Creating a Redis OSS (cluster mode enabled) cluster (Console)

Enabling At-Rest Encryption Using the AWS CLI

To enable at-rest encryption when creating a 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 Redis OSS (Cluster Mode Disabled) Cluster (CLI)

The following operation creates the 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 `redis`.
- **`--engine-version`**—Must be 3.2.6, 4.0.10 or later.
- **`--at-rest-encryption-enabled`**—Required to enable at-rest encryption.

Example 1: 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 Redis OSS (Cluster Mode Disabled) replication group from scratch (AWS CLI)
- create-replication-group

Enabling At-Rest Encryption on a Cluster for Redis OSS (Cluster Mode Enabled) (CLI)

The following operation creates the 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 `redis`.
- **`--engine-version`**—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 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.m3.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.m3.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 Redis OSS (Cluster Mode Enabled) replication group from scratch (AWS CLI)
- create-replication-group

See Also

- Amazon VPCs and ElastiCache security
- Identity and Access Management for Amazon ElastiCache

Authentication and Authorization

ElastiCache supports authenticating users using IAM and the Redis OSS AUTH command, and authorizing user operations using Role-Based Access Control (RBAC).

Topics

- Role-Based Access Control (RBAC)
- Authenticating with the Redis OSS AUTH command
- Disabling access control on an ElastiCache Redis OSS cache

Role-Based Access Control (RBAC)

Instead of authenticating users with the Redis OSS AUTH command as described in Authenticating with the Redis OSS AUTH command, in Redis OSS 6.0 onward you can use a feature called Role-Based Access Control (RBAC). RBAC is also the only way to control access to serverless caches.

Unlike Redis OSS AUTH, where all authenticated clients have full cache access if their token is authenticated, RBAC enables you to control cache access through user groups. These user groups 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 user groups aligned with a specific role (administrators, human resources) that are then deployed to one or more ElastiCache (Redis OSS) caches. By doing this, you can establish security boundaries between clients using the same Redis OSS cache or caches and prevent clients from accessing each other's data.

RBAC is designed to support the introduction of Redis OSS ACL in Redis OSS 6. When you use RBAC with your ElastiCache (Redis OSS) cache, there are some limitations:

- You can't specify passwords in an access string. You set passwords with CreateUser or ModifyUser calls.
- For user rights, you pass 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. 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 (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 (Redis OSS) supports the `ACL LIST` command, but does not include support for password hashes as Redis OSS does. With ElastiCache (Redis OSS), you can use the describe-users operation to get similar information, including the rules contained within the access string. However, describe-users doesn't retrieve a user password.

  Other read-only commands supported by ElastiCache (Redis OSS) include ACL WHOAMI, ACL USERS, and ACL CAT. ElastiCache (Redis OSS) doesn't support any other write-based ACL commands.
- The following constraints apply:

Resource	Maximum allowed
Users per user group	100
Number of users	1000

Resource	Maximum allowed
Number of user groups	100

Using RBAC with ElastiCache (Redis OSS) is described in more detail following.

Topics

Specifying Permissions Using an Access String

To specify permissions to an ElastiCache (Redis OSS) cache, you create an access string and assign it to a user, using 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 key permissions for more information.
- `%RW~<pattern>` – Alia 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.

Applying RBAC to a Cache for ElastiCache (Redis OSS)

To use ElastiCache (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.

These steps are described in detail following.

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.

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** and for select the user that you want as the default 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 "REDIS" \
    --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 "REDIS" ^
    --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 "REDIS" \
  --user-ids "new-default-user"
```

   For Windows:

```
aws elasticache create-user-group ^
  --user-group-id "new-group-2" ^
  --engine "REDIS" ^
  --user-ids "new-default-user"
```

3. Swap the new `default` user with the original `default` user.

   For Linux, macOS, or Unix:

```
aws elasticache modify-user-group \
    --user-group-id test-group \
    --user-ids-to-add "new-default-user" \
    --user-ids-to-remove "default"
```

   For Windows:

```
aws elasticache modify-user-group ^
    --user-group-id test-group ^
    --user-ids-to-add "new-default-user" ^
    --user-ids-to-remove "default"
```

   When this modify operation is called, any existing connections to a cache that the original default user has are terminated.

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 (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:*" \
  --no-password-required
```

For Windows:

```
aws elasticache modify-user ^
  --user-id user-id-1 ^
  --access-string "~objects:* ~items:* ~public:*" ^
  --no-password-required
```

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**
     >
     > When you create a user group, you are required to include the default user.

   - **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 "REDIS" \
    --user-ids user-id-1, user-id-2
  ```

  For Windows:

  ```
  aws elasticache create-user-group ^
    --user-group-id "new-group-1" ^
    --engine "REDIS" ^
    --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 userid-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 Redis OSS (cluster mode disabled) cluster (Console)
- For cluster mode enabled, see Creating a 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 `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 "redis" \
    --user-group-id "new-group-1"
```

For Windows:

```
aws elasticache create-serverless-cache ^
    --serverless-cache-name "new-serverless-cache" ^
    --description "new-serverless-cache" ^
    --engine "redis" ^
    --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 Redis OSS (cluster mode disabled) cluster (Console)
- For cluster mode enabled, see Creating a 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 with encryption in transit (TLS) enabled and the **user-group-ids** 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 `redis`.
- **`--engine-version`** – Must be 6.0 or later.
- **`--transit-encryption-enabled`** – Required 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 "redis" \
    --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 "redis" ^
    --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 with encryption in transit (TLS) enabled and the **user-group-ids** parameter with the value *my-user-group-id*.

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 Redis OSS AUTH to RBAC

If you are using Redis OSS AUTH as described in Authenticating with the Redis OSS AUTH command and want to migrate to using RBAC, use the following procedures.

Use the following procedure to migrate from Redis OSS AUTH to RBAC using the console.

To migrate from 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 running 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 Redis OSS AUTH to RBAC using the CLI.

To migrate from Redis OSS 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 Redis OSS AUTH

If you are using RBAC and want to migrate to Redis OSS AUTH , see Migrating from RBAC to Redis OSS 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 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 (Redis OSS) passwords (that is, secrets) using an AWS Lambda function that Secrets Manager provides.

For more information about AWS Secrets Manager, see What is AWS Secrets Manager?

How ElastiCache uses secrets

With Redis OSS 6, ElastiCache (Redis OSS) introduced Role-Based Access Control (RBAC) to secure the 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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Action": [
            "elasticache:DescribeUsers",
            "elasticache:ModifyUser"
        ],
        "Resource": "arn:aws:elasticache:us-east-1:xxxxxxxxxx918:user:user1"
        }
    ]
}
```

Secrets Manager

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:xxxxxxxxxxx: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

Topics

- Overview
- Limitations
- Setup
- Connecting

Overview

With IAM Authentication you can authenticate a connection to ElastiCache (Redis OSS) using AWS IAM identities, when your cache is configured to use 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 (Redis OSS) works by providing a short-lived IAM authentication token instead of a long-lived ElastiCache user password in the 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 Redis OSS access. You can also grant access to users from their federated Identity providers directly to Redis OSS caches.

To use AWS IAM with ElastiCache (Redis OSS), 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 Redis OSS Client when connecting to your Redis OSS cache. A Redis OSS client with support for credentials provider can auto-generate the temporary credentials automatically for each new connection. ElastiCache for Redis OSS 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 Redis OSS version 7.0 or above.
- For IAM-enabled ElastiCache users the username and user id properties must be identical.
- The IAM authentication token is valid for 15 minutes. For long-lived connections, we recommend using a Redis OSS client that supports a credentials provider interface.
- An IAM authenticated connection to ElastiCache for 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 authentication is not supported in `MULTI EXEC` commands.
- Currently, IAM authentication supports the following global condition context keys:
  - When using IAM authentication with serverless caches, `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.

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*.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": { "AWS": "arn:aws:iam::123456789012:root" },
        "Action": "sts:AssumeRole"
    }
}
```

3. 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"
      ]
    }
  ]
}
```

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
```

Connecting

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 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 = new
 DefaultAWSCredentialsProviderChain();

// 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
 (Redis OSS).
IAMAuthTokenRequest iamAuthTokenRequest = new IAMAuthTokenRequest(userId, cacheName,
 region, isServerless);
String iamAuthToken =
 iamAuthTokenRequest.toSignedRequestUri(awsCredentialsProvider.getCredentials());

// 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 HttpMethodName REQUEST_METHOD = HttpMethodName.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 long TOKEN_EXPIRY_SECONDS = 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) throws
 URISyntaxException {
        Request<Void> request = getSignableRequest();
        sign(request, credentials);
        return new URIBuilder(request.getEndpoint())
            .addParameters(toNamedValuePair(request.getParameters()))
            .build()
            .toString()
            .replace(REQUEST_PROTOCOL, "");
    }

    private <T> Request<T> getSignableRequest() {
        Request<T> request  = new DefaultRequest<>(SERVICE_NAME);
        request.setHttpMethod(REQUEST_METHOD);
        request.setEndpoint(getRequestUri());
        request.addParameters(PARAM_ACTION, Collections.singletonList(ACTION_NAME));
        request.addParameters(PARAM_USER, Collections.singletonList(userId));
        if (isServerless) {
            request.addParameters(PARAM_RESOURCE_TYPE,
 Collections.singletonList(RESOURCE_TYPE_SERVERLESS_CACHE));
        }
        return request;
    }

    private URI getRequestUri() {
        return URI.create(String.format("%s%s/", REQUEST_PROTOCOL, cacheName));
    }

    private <T> void sign(SignableRequest<T> request, AWSCredentials credentials) {
        AWS4Signer signer = new AWS4Signer();
        signer.setRegionName(region);
        signer.setServiceName(SERVICE_NAME);

        DateTime dateTime = DateTime.now();
        dateTime = dateTime.plus(Duration.standardSeconds(TOKEN_EXPIRY_SECONDS));
```

```
        signer.presignRequest(request, credentials, dateTime.toDate());
    }

    private static List<NameValuePair> toNamedValuePair(Map<String, List<String>> in) {
        return in.entrySet().stream()
            .map(e -> new BasicNameValuePair(e.getKey(), e.getValue().get(0)))
            .collect(Collectors.toList());
    }
}
```

Connect with credentials provider

The code below shows how to authenticate with ElastiCache for Redis OSS 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 = new
 DefaultAWSCredentialsProviderChain();

// Create an IAM authentication token request. Once this request is signed it can be
 used as an
// IAM authentication token for ElastiCache (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.userName = userName;
        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.getCredentials());
    }
}
```

Authenticating with the Redis OSS AUTH command

> **Note**
>
> Redis OSS **AUTH** has been superseded by the section called “Role-Based Access Control (RBAC)”. All serverless caches must use RBAC for authentication.

Redis OSS authentication tokens or passwords enable Redis OSS to require a password before allowing clients to run commands, thereby improving data security. Redis OSS **AUTH** is available for self-designed clusters only.

Topics

Overview of AUTH in ElastiCache (Redis OSS)

When you use the Redis OSS **AUTH** with your ElastiCache (Redis OSS) cluster, there are some refinements.

In particular, be aware of these AUTH token or password constraints when using the AUTH with ElastiCache (Redis OSS):

- Tokens, or passwords, must be 16–128 printable characters.
- Nonalphanumeric characters are restricted to (!, &, #, $, ^, <, >, -).
- AUTH can only be enabled for encryption in-transit enabled ElastiCache (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 (Redis OSS) cluster

You can require that users enter a token (password) on a token-protected 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 with encryption in transit (TLS) enabled and 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 `redis`.
- **`--engine-version`** – Must be 3.2.6, 4.0.10, or later.
- **`--transit-encryption-enabled`** – Required for authentication and HIPAA eligibility.
- **`--auth-token`** – Required for HIPAA eligibility. This value must be the correct token for this token-protected 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 ElastiCache (Redis OSS) cluster

To make it easier to update your authentication, you can modify the **AUTH** token used on an ElastiCache (Redis OSS) cluster. You can make this modification if the engine version is 5.0.6 or higher and if ElastiCache (Redis OSS) has encryption in transit enabled.

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 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 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 Redis OSS 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 ElastiCache (Redis OSS) cluster

To enable authentication on an existing 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 Redis OSS with the AUTH token, use the SET strategy to mark the AUTH token as required. Enabling authentication is only supported on Redis OSS servers with encryption in transit (TLS) enabled.

Migrating from RBAC to Redis OSS AUTH

If you are authenticating users with Redis OSS Role-Based Access Control (RBAC) as described in [Role-Based Access Control (RBAC)](), and you want to migrate to Redis OSS AUTH, use the following procedures. You can migrate using either console or CLI.

To migrate from RBAC to Redis OSS 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 **Redis OSS AUTH default user access**.
7. Under **Redis OSS AUTH token**, set a new token.
8. Choose **Preview changes** and then on the next screen, **Modify**.

To migrate from RBAC to Redis OSS AUTH using the AWS CLI

Use one of the following commands to configure a new optional **AUTH** token for your 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 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 Redis OSS 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 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 Redis OSS cache".

Disabling access control on an ElastiCache Redis OSS cache

Follow the instructions below to disable access control on a Redis OSS TLS-enabled cache. Your Redis OSS cache will have one of two different types of configurations: Redis OSS 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.

To disable a 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 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 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.

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 current generation nodes are launched in Amazon Virtual Private Cloud only.

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 cache cluster to the virtual network, and control access to the cache 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

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.

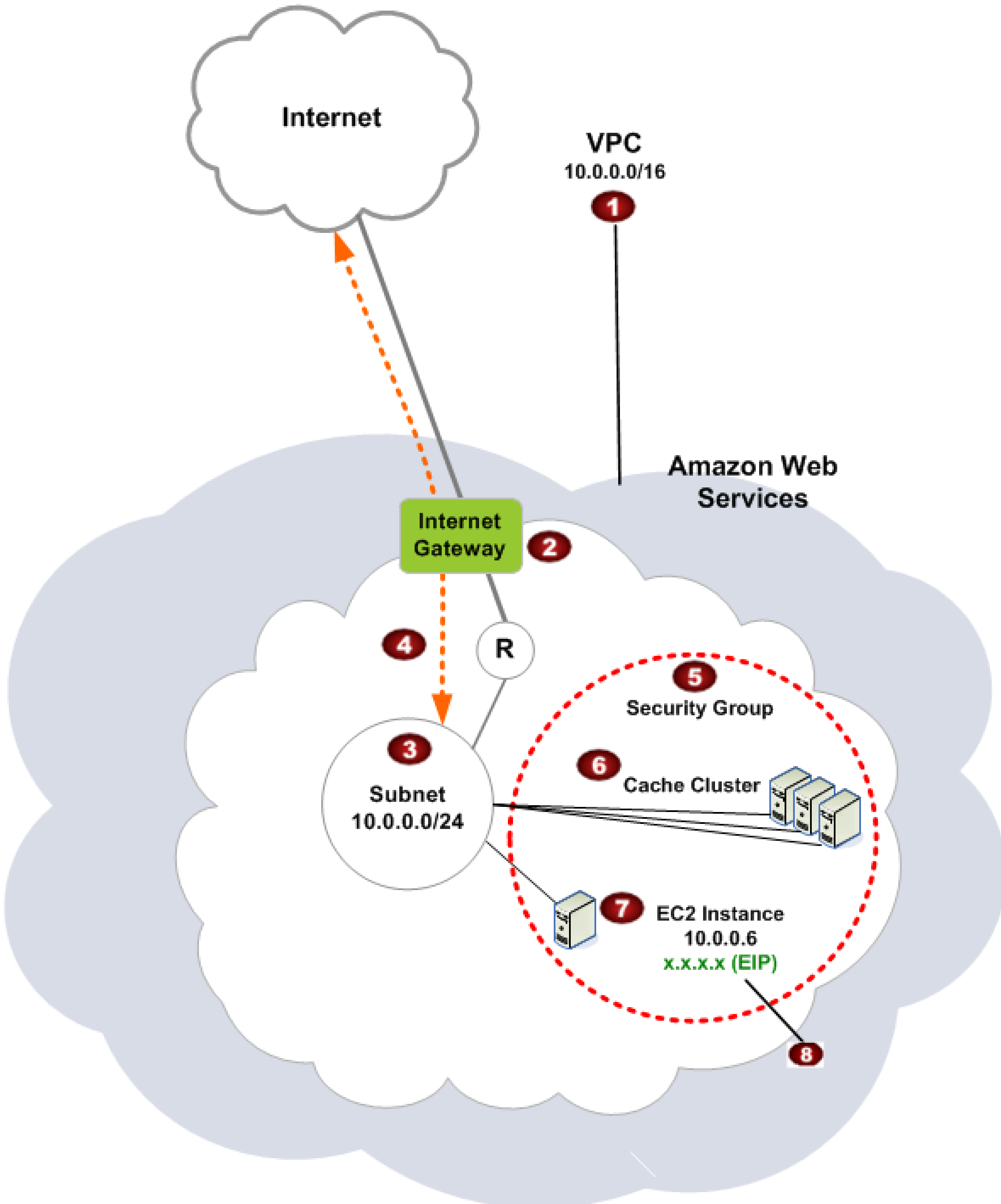
With Amazon Virtual Private Cloud, you can create a virtual network in the AWS cloud that closely resembles a traditional data center. You can configure your Amazon VPC, including selecting its IP address range, creating subnets, and configuring route tables, network gateways, and security settings.

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 ElastiCache 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 API 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

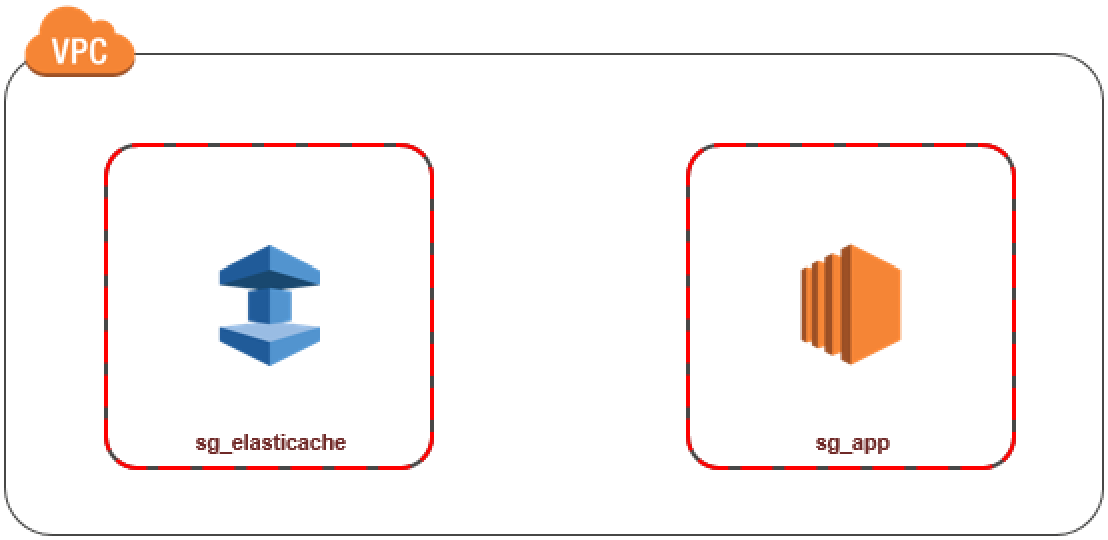
Amazon ElastiCache supports the following scenarios for accessing a cache in an Amazon VPC:

Contents

Accessing an ElastiCache Cache when it and the Amazon EC2 Instance are in the Same Amazon VPC

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 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.

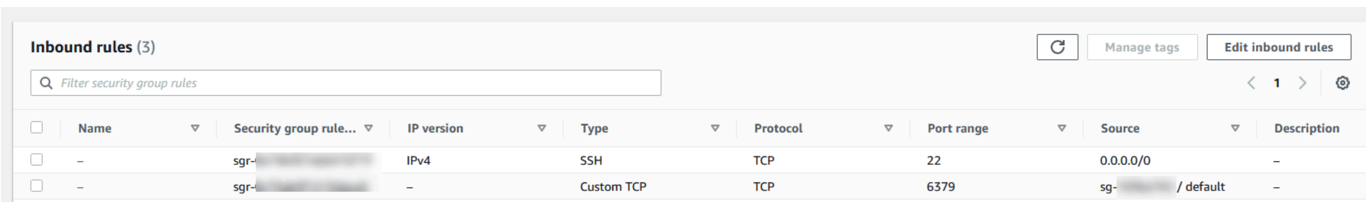
To create a rule in a VPC security group that allows connections from another security group

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 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

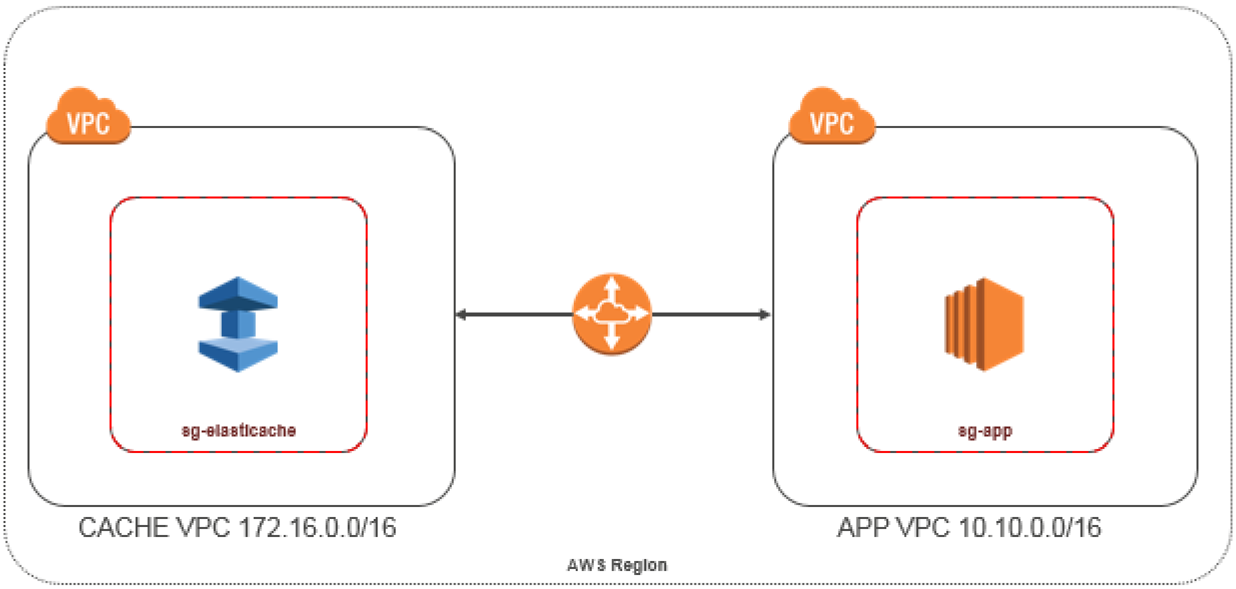
When your cache is in a different VPC from the EC2 instance you are using to access it, there are several ways to access the cache. If the cache and EC2 instance are in different VPCs but in the same region, you can use VPC peering. If the cache and the EC2 instance are in different regions, you can 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 enable multicast on a transit gateway, and then create a transit gateway multicast domain that allows multicast traffic to be sent from your multicast source to multicast group members over VPC attachments that you associate with the domain.

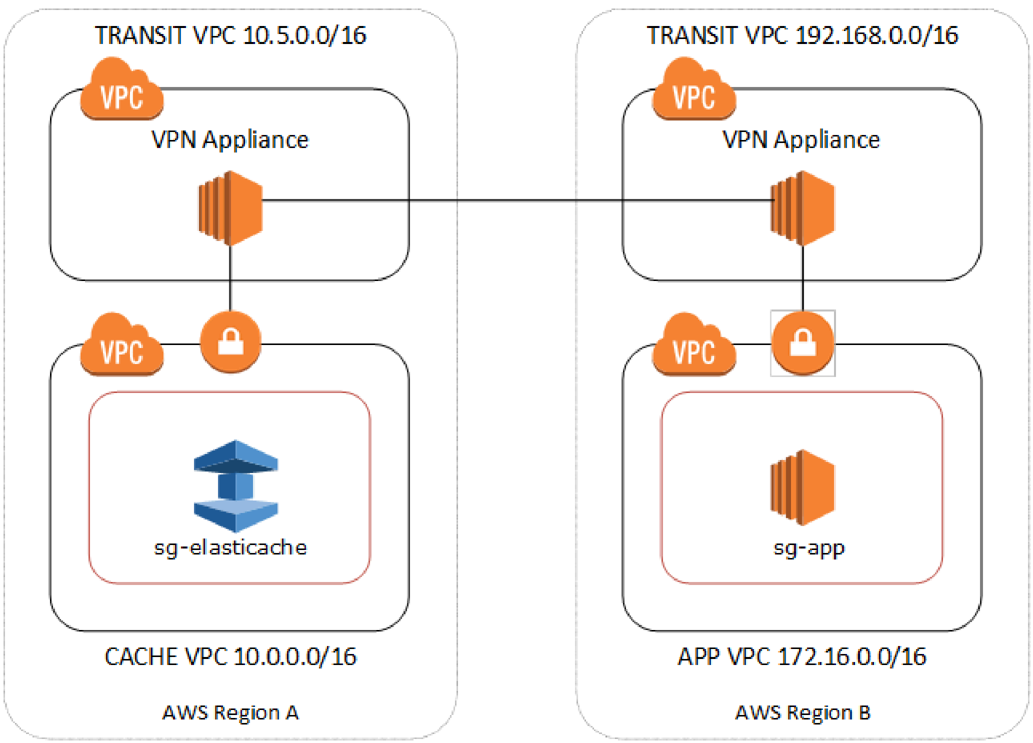
You can also create a peering connection attachment between transit gateways in different AWS Regions. This enables you to route traffic between the transit gateways' attachments across different Regions.

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

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.

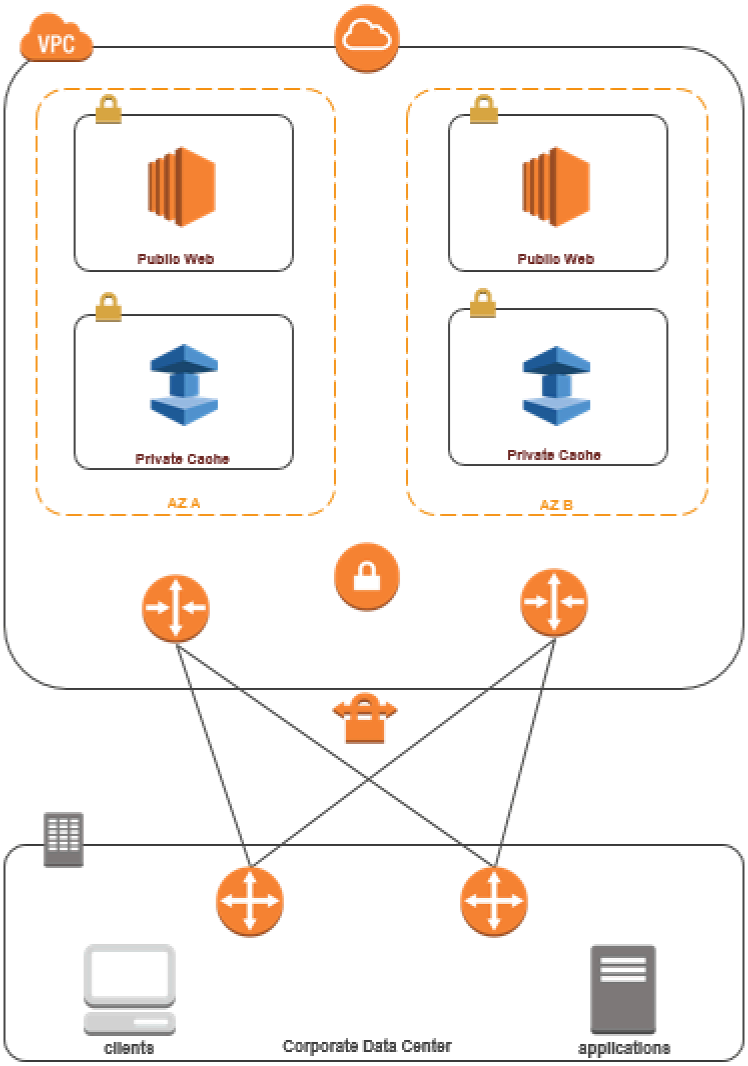
Accessing an ElastiCache Cache from an Application Running in a Customer's Data Center

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

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

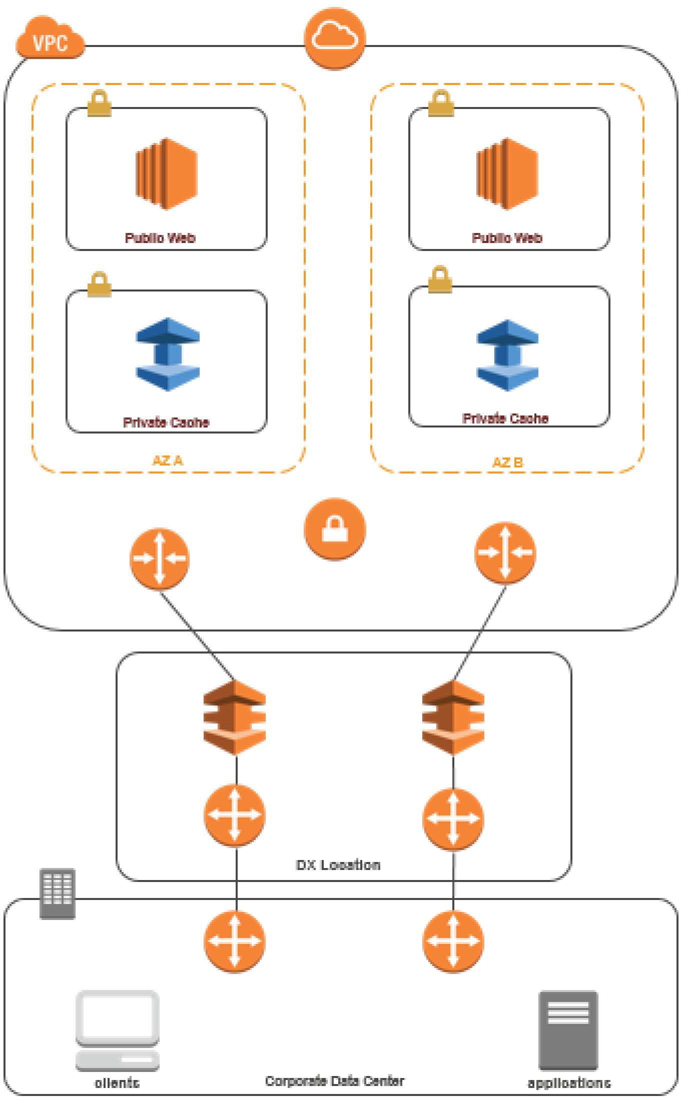
To access a cache in a VPC from on-prem application over VPN connection

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.

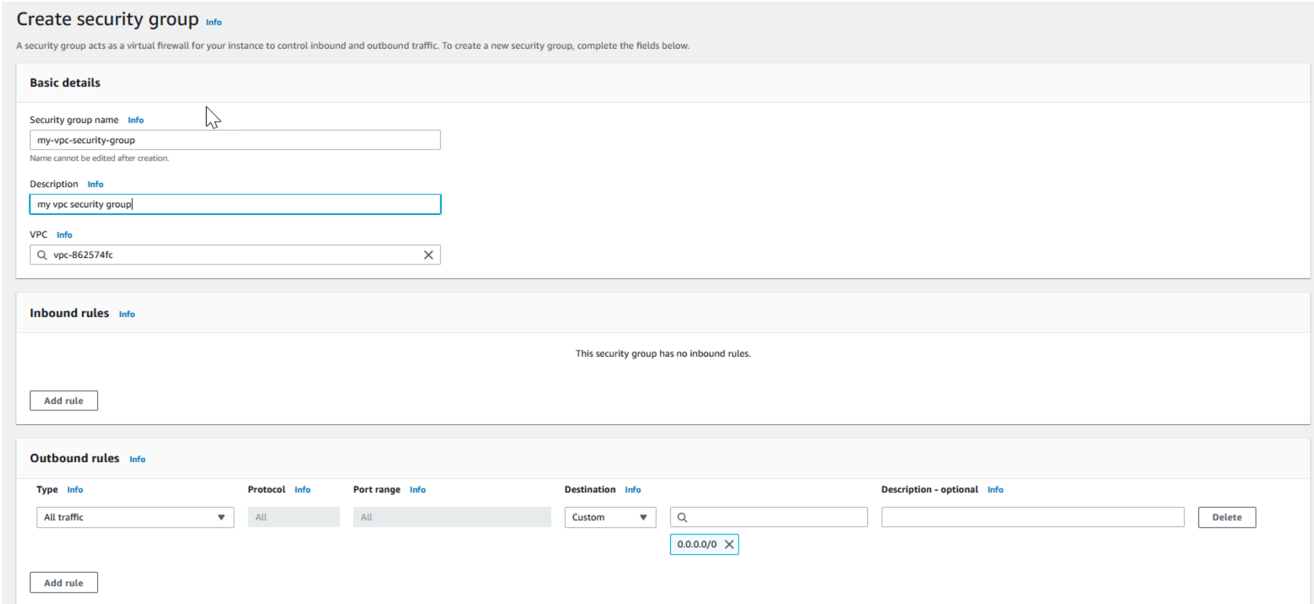
> **Tip**
>
> Make a note of your subnet identifiers, and which are public and private. You will need this information later when you launch your clusters and add an Amazon EC2 instance to your Amazon VPC.

7. Create an Amazon VPC security group. You will use this group for your cache 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 cache cluster in your Amazon VPC.

- Creating a subnet group
- Creating a Redis OSS (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.

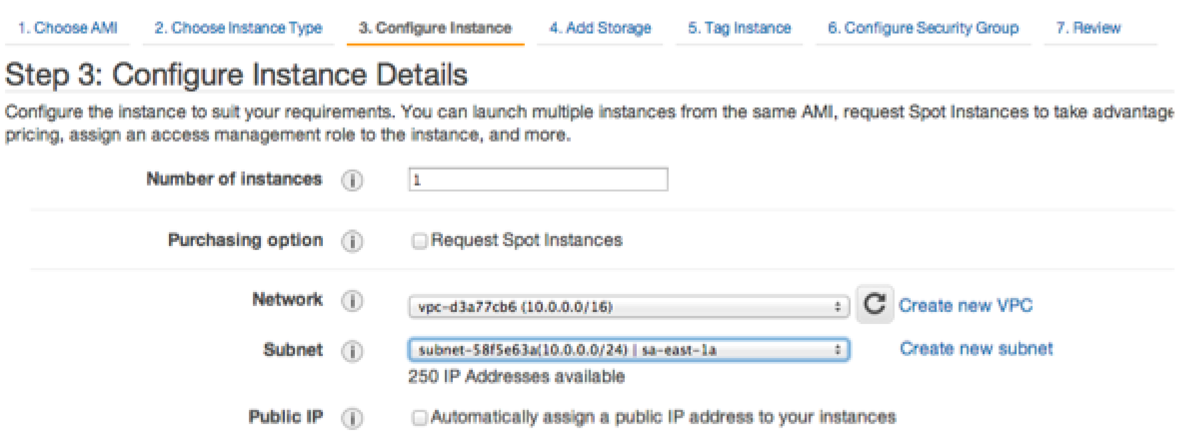
> **Note**
>
> For information about using Amazon EC2, see the [Amazon EC2 Getting Started Guide] in the [Amazon EC2 documentation].

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:

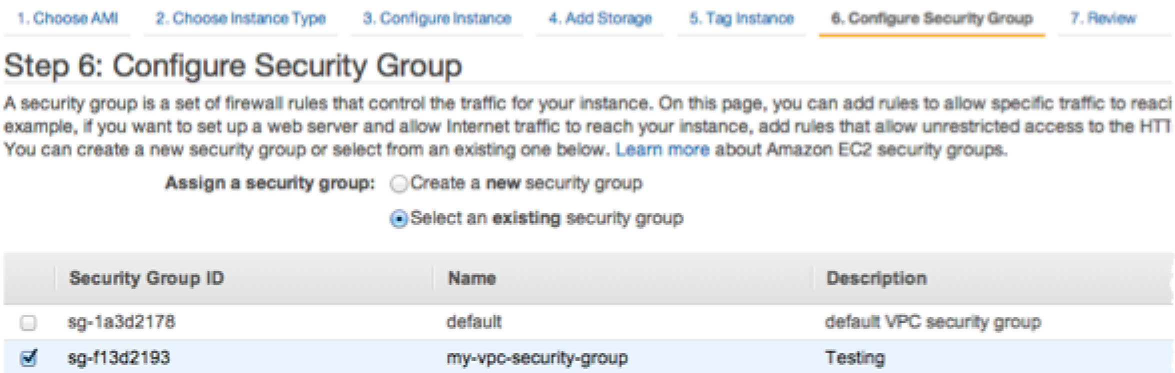
    a. In the **Network** list, choose your Amazon VPC.

    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 cache cluster (AWS CLI)

1. 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!
```

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 cache 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.

Amazon 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) resolves 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 it.

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) resolves 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` for Beijing and `elasticache.cn-northwest-1.amazonaws.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

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 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 self-designed clusters running in an Amazon Virtual Private Cloud (VPC) environment.

If you create a self-designed 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 cluster or replication group.

Topics

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 Network type. 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.

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.

- **Standalone Redis OSS cluster –** To launch a single-node Redis OSS cluster, see Creating a Redis OSS (cluster mode disabled) cluster (Console). In step 7.a (**Advanced Redis OSS Settings**), choose a VPC subnet group.
- **Redis OSS (cluster mode disabled) replication group –** To launch a Redis OSS (cluster mode disabled) replication group in a VPC, see Creating a Redis OSS (Cluster Mode Disabled) replication group from scratch. In step 7.b (**Advanced Redis OSS Settings**), choose a VPC subnet group.
- **Redis OSS (cluster mode enabled) replication group –** Creating a 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.

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

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in ElastiCache.

Service user – If you use the ElastiCache service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more ElastiCache features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in ElastiCache, see Troubleshooting Amazon ElastiCache identity and access.

Service administrator – If you're in charge of ElastiCache resources at your company, you probably have full access to ElastiCache. It's your job to determine which ElastiCache features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with ElastiCache, see How Amazon ElastiCache works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to ElastiCache. To view example ElastiCache identity-based policies that you can use in IAM, 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* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

  - **Service role –** A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role –** A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) 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 is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

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 Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing 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 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.

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.

Access control lists (ACLs)

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.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries –** A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies –** Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. 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 Creating IAM 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. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. 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. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. 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. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

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. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, 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 (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. 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.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. 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. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

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 What is ABAC? 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

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM] in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role (console)] in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM].

Cross-service principal permissions for ElastiCache

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. 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 [Creating 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. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies 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 AWS 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 IAM Access Analyzer policy validation 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 Configuring MFA-protected API access 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

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 Creating 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 Creating 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

- 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 cache cluster.
- Suppose that you create an IAM user in your AWS account and grant permissions to create a cache cluster to that user. In this case, the user can create a cache cluster. However, your AWS account, to which the user belongs, owns the cache cluster resource.
- Suppose that you create an IAM role in your AWS account with permissions to create a cache cluster. In this case, anyone who can assume the role can create a cache 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)

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 cache 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).

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "DescribeCacheClusters",
      "Effect": "Allow",
      "Action": [
         "elasticache:DescribeCacheClusters"],
      "Resource": resource-arn
      }
   ]
}
```

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 self-designed 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 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.

```
{
    "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 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 cache 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.

```
{
   "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.

```
{
   "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.

```
{
   "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.

```
{
  "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.

```
{
  "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.

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.

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 self-designed cache cluster created will be of node type `cache.r5.large`.

```
{
    "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"
```

```
                    ]
                }
            }
        }
    ]
}
```

For more information, see Tag-Based access control policy examples.

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.

```
{
    "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.

```
{
    "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.

```
{
    "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:NumNodeGroups**: Create a replication group with fewer than 20 node groups.

```
{
    "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"
                }
            }
        }
    ]
}
```

5. **elasticache:ReplicasPerNodeGroup**: Specify the replicas per node between 5 and 10.

```
{
    "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"
                }
            }
        }
    ]
}
```

6. **elasticache:EngineVersion**: Specify usage of engine version 5.0.6.

```
{
    "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"
                }
            }
        }
    ]
}
```

7. **elasticache:EngineType**: Specify using Redis OSS engine only.

```
{
    "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"
                }
            }
        }
    ]
}
```

8. **elasticache:AtRestEncryptionEnabled**: Specify that replication groups would be created only with encryption enabled.

```
{
    "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"
                }
```

```
            }
        }
    ]
}
```

9. **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:

```
{
    "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:

```
{
    "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:

```
{
    "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.

10. **elasticache:AutomaticFailoverEnabled**: Specify that replication groups would be created only with automatic failover enabled.

```
{
    "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"
                }
            }
        }
    ]
}
```

11. **elasticache:MultiAZEnabled**: Specify that replication groups cannot be created with Multi-AZ disabled.

```
{
    "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"
                }
            }
        }
    ]
}
```

12. **elasticache:ClusterModeEnabled**: Specify that replication groups can only be created with cluster mode enabled.

```
{
    "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"
                }
            }
        }
    ]
}
```

13. **elasticache:AuthTokenEnabled**: Specify that replication groups can only be created with AUTH token enabled.

```
{
    "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"
                }
            }
        }
    ]
}
```

14. **elasticache:SnapshotRetentionLimit**: Specify the number of days (or min/max) to keep the snapshot. Below policy enforces storing backups for at least 30 days.

```
{
    "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"
                }
            }
```

```
        }
    ]
}
```

15. **elasticache:KmsKeyId**: Specify usage of customer managed AWS KMS keys.

```
{
  "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"
            }
        }
    }
  ]
}
```

16. **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".

```
{
    "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"
                }
            }
        }
    ]
}
```

17. **elasticache:CreateCacheCluster**: Denying `CreateCacheCluster` action if the request tag `Project` is missing or is not equal to `Dev`, `QA` or `Prod`.

```
{
    "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"
                    ]
                }
            }
        }
    ]
```

```
}
```

18. **elasticache:CacheNodeType**: Allowing `CreateCacheCluster` with `cacheNodeType` cache.r5.large or cache.r6g.4xlarge and tag `Project=XYZ`.

```
{
  "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"
        }
      }
    }
  ]
}
```

> **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

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
  - 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.

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. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

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 Redis OSS
- Recommendations

Mitigating Failures when Running Redis OSS

When running the 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. Self-designed clusters must be configured appropriately to mitigate the failure of an individual node.

To mitigate the impact of Redis OSS node failures on self-designed clusters, you have the following options:

Topics

Mitigating Failures: Redis OSS Replication Groups

A 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.

Redis OSS Multi-AZ

You can enable Multi-AZ on your Redis OSS replication groups. Whether you enable Multi-AZ or not, a failed primary will be detected and replaced automatically. How this takes place varies whether or not Multi-AZ is or is not enabled.

When Multi-AZ is enabled

1. ElastiCache detects the primary node failure.
2. ElastiCache 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, see Minimizing downtime in ElastiCache (Redis OSS) with Multi-AZ.

When Multi-AZ is disabled

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. ElastiCache syncs the new primary with one of the existing replicas.
5. When the sync is finished, the new node functions as the cluster's primary node.

During steps 1 through 4 of this process, your application can't write to the primary node. However, your application can continue reading from your replica nodes.

For added protection, we recommend that you launch the nodes in your replication group in different Availability Zones (AZs). 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 self-designed 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 Redis OSS nodes, see Creating a Redis OSS (cluster mode disabled) cluster (Console).

For more information on regions and Availability Zones, see Choosing regions and availability zones.

Recommendations

We recommend creating serverless caches over self-designed clusters, as you automatically obtain better fault tolerance without additional configuration. When creating a self-designed 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, 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 self-designed clusters, 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.

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 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 self-designed clusters, you set up a predefined maintenance window so that ElastiCache can apply these updates. However, in some cases you might find this approach too rigid and likely to constrain your business flows.

With service updates, you control when and which updates are applied to your self-designed clusters. You can also monitor the progress of these updates to your selected ElastiCache cluster in real time.

Managing service updates

ElastiCache service updates for self-designed clusters are released on a regular basis. If you have one or more qualifying self-designed 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. We strongly recommend that you apply any updates of type **security-update** as soon as possible to ensure that your ElastiCache clusters are always up-to-date with current security patches.

The following sections explore these options in detail.

Topics

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 not take 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** or **engine-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.
   - **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 clusters, run the following command:

  ```
  aws elasticache batch-apply-update-action --service-update 
  ServiceUpdateNameToApply=sample-service-update --cluster-names cluster-1 
  cluster2
  ```

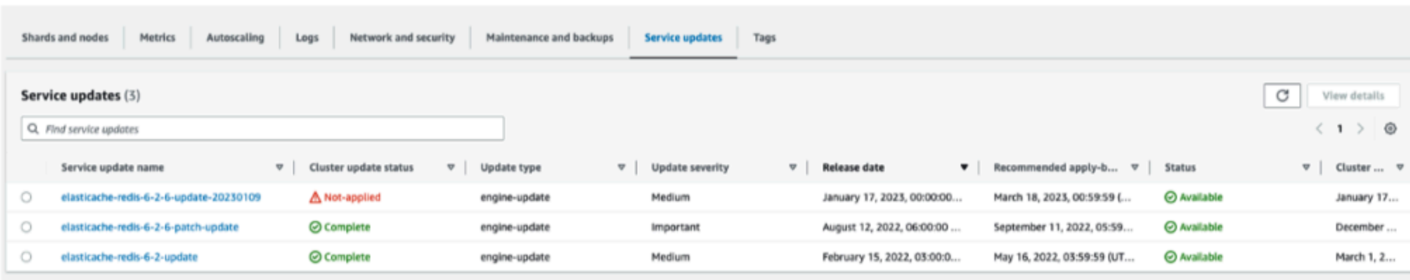
  For more information, see batch-apply-update-action.

Verifying you have the latest Service Update Applied using the AWS console

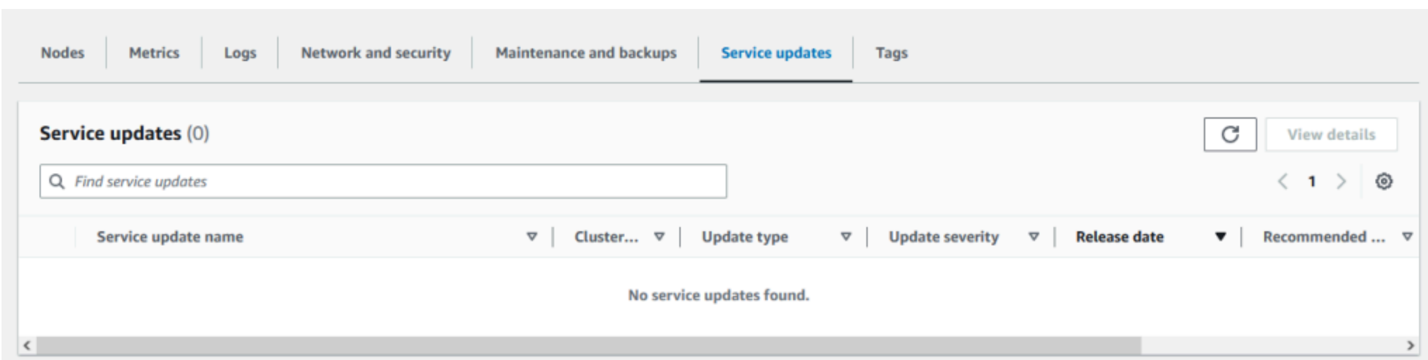
You can verify your ElastiCache (Redis OSS) clusters are running the latest service update by following these steps:

1. Choose an applicable cluster on the **Redis Clusters** 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 (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 sample-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 sample-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 (Redis OSS)

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 (Redis OSS).

We recommend that you always upgrade to the latest ElastiCache (Redis OSS) version to be protected against known vulnerabilities. When operating an ElastiCache Serverless Cache, CVE fixes are automatically applied to your cache. When operating self-designed clusters, ElastiCache (Redis OSS) exposes the PATCH component. For example, when using ElastiCache (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.

You can use this page to verify whether a particular version of ElastiCache (Redis OSS) has a fix for a specific security vulnerability. If your ElastiCache (Redis OSS) cluster is running a version without the security fix, refer to the table below and take action. You can either upgrade to a more recent ElastiCache (Redis OSS) version containing the fix, or if you are on an ElastiCache (Redis OSS) version containing the fix, ensure you have the latest service update applied by referring to Managing service updates. For more information on the supported ElastiCache (Redis OSS) engine versions and how to upgrade, see Engine versions and upgrading.

Note

- If a CVE is addressed in an ElastiCache (Redis OSS) version, it means it is also addressed in the newer versions. So for example if a vulnerability is addressed in ElastiCache (Redis OSS) Version 6.0.5, this continues forward for Versions 6.2.6, 7.0.7, and 7.1.
- An asterisk (*) in the following table indicates you must have the latest service update applied for the ElastiCache (Redis OSS) Cluster running the ElastiCache (Redis OSS) Version specified in order to address the security vulnerability. For more information on how to verify you have the latest service update applied for the ElastiCache (Redis OSS) version your cluster is running on, see Managing service updates.

ElastiCache (Redis OSS) version	CVEs Addressed
Redis OSS 6.0.5	CVE-2022-24735*, CVE-2022-24736*
Redis OSS 6.2.6	CVE-2022-24834*, CVE-2022-35977*, CVE-2022-36021*, CVE-2022-24735, CVE-2022-24736
Redis OSS 7.0.7	CVE-2023-41056*, CVE-2022-24834*, CVE-2022-35977, CVE-2022-36021, CVE-2022-24735, CVE-2022-24736
Redis OSS 7.1.0	CVE-2023-41056, CVE-2022-24834, CVE-2022-35977, CVE-2022-36021, CVE-2022-24735, CVE-2022-24736

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

Serverless metrics and events

This section describes the metrics and events that you can monitor when working with serverless caches.

Topics

Serverless cache metrics

The `AWS/ElastiCache` namespace includes the following CloudWatch metrics for your Redis OSS 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)	Count

Metric	Description	Unit
	consumed by the requests executed on your cache	
SuccessfulReadRequestLatency	Latency of successful read requests.	Microseconds
SuccessfulWriteRequestLatency	Latency of successful write requests	Microseconds
TotalCmdsCount	Total count of all commands executed on your cache	Count
CacheHitRate	Indicates the hit rate of your cache. This is calculated using `cache_hits` and `cache_misses` statistics in the following way: `cache_hits /(cache_hits + cache_misses)` .	Percent
CacheHits	The number of successful 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 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

Metric	Description	Unit
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
IamAuthenticationExpirations	The total number of expired IAM-authenticated Redis OSS connections. You can find more information about Authenticating with IAM in the user guide.	Count
IamAuthenticationThrottling	The total number of throttled IAM-authenticated Redis OSS AUTH or HELLO requests. You can find more information about Authenticating with IAM 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

Metric	Description	Unit
AuthenticationFailures	The total number of failed attempts to authenticate to Redis OSS using the AUTH command. We suggest setting an alarm on this to detect unauthorized access attempts.	Count
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 Redis OSS commandstats statistic. It's derived by summing all of	Count

Metric	Description	Unit
	the geo type of commands: geoadd, geodist, geohash, geopos, georadius, and georadiusbymember.	
GeoSpatialBasedCmdsECPUs	ECPUs consumed by geospatial-based commands.	Count
GetTypeCmds	The total number of read-only type commands. This is derived from the 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 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

Metric	Description	Unit
HyperLogLogBasedCmds	The total number of HyperLogLog-based commands. This is derived from the Redis OSS commandstats statistic by summing all of the pf type of commands (pfadd, pfcount, pfmerge, and so on.).	Count
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 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 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

Metric	Description	Unit
JsonBasedSetCmds	The total number of JSON write commands. This is derived from the Redis OSS commandstats statistic by summing all JSON write commands that act upon JSON keys.	Count
JsonBasedSetCmdsECPUs	ECPUs consumed by JSON write commands.	Count
KeyBasedCmds	The total number of commands that are key-based . This is derived from the 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-base d. This is derived from the 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 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 Redis OSS commandstatsstatis tics by summing all of the commands used for pub/ sub functionality: psubscrib e, publish, pubsub, punsubscr ibe, ssubscribe, sunsubscr ibe, spublish, subscribe, and unsubscribe.	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 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 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 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 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-ba sed commands.	Count
StreamBasedCmds	The total number of commands that are stream-based. This is derived from the 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 https://docs.aws.amazon.com/eventbridge/latest/userguide/.

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 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 permissions to create a VPC endpoint.
Cache creation failed	Cache arn	failure	serverless-cache	Failed to create cache <cache-name>. A user with an incompatible Redis OSS version is present in user group <user-group-name>.
Cache creation failed	Cache arn Cache snapshot arn	failure	serverless-cache	Failed to create cache <cache-name>. The provided user group <user-group-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>. Failure reasons: • failed to retrieve file from S3. • expected md5 does not match actual md5. • the provided RDB file has an unsupported version.

Serverless Cache Update Events

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>.

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 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

Detail-Type	Resources list	Category	Source	Message
Cache deleted	Cache arn	deletion	serverless-cache	Cache <cache-name> was deleted.

Serverless Cache Usage Limit Events

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 configura tion.
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

Detail-Type	Resources-list	Category	Source	Message
Snapshot created	Cache arn Snapshot arn	creation	serverless-cache-snapshot	Snapshot <snapshot-name> created for cache <cache-name>.
Snapshot creation failed	Cache arn Snapshot arn	failure	serverless-cache-snapshot	Failed to create snapshot for cache <cache-na me>. Snapshot <snapshot-name> creation failed with Customer Managed

Detail-Type	Resources-list	Category	Source	Message
				Key <key-id> <reason>. Failure reason messages: • because Customer Managed Key is disabled • because Customer Managed Key cannot be found • because the request timed out
Snapshot creation failed	Cache arn Snapshot arn	failure	serverless-cache-snapshot	Failed to create snapshot for cache <cache-name>. Snapshot <snapshot-name> creation failed because <reason>. Default reason: • 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 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 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'.

Self-designed clusters metrics and events

This section describes the metrics, events, and logs that you can expect to see when you work with self-designed clusters.

Topics

Metrics for self-designed clusters

When you self-design clusters, 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 Redis OSS.

Events for self-designed clusters

ElastiCache logs events that relate to your self-designed caches. When working with self-designed clusters, you can view your cluster events in the ElastiCache console, using the AWS CLI, or using Amazon Simple Notification Service (SNS). Self-designed cluster events are not published to Amazon EventBridge.

Self-designed 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 self-designed cluster events.

```
aws elasticache describe-events --source-type cache-cluster --max-items 40
```

The following code lists all events for self-designed caches for the past 24 hours (1440 minutes).

```
aws elasticache describe-events --source-type cache-cluster --duration 1440
```

Self-designed cluster events

This section contains the list of events you can expect to receive for your self-designed 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 cache 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 cache cluster parameters have been changed.
ElastiCache:CacheClusterProvisioningComplete	`ElastiCache:CacheClusterProvisioningComplete `*`cluster-name-0001-005`*	The provisioning of a cache 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 cache 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.

Event Name	Message	Description
ElastiCache:CacheClusterSecurityGroupModified	`ElastiCache:CacheClusterSecurityGroupModified : ` *`cluster-name`*	One of the following events has occurred: • The list of cache security groups authorized for the cache cluster has been modified. • One or more new EC2 security groups have been authorized on any of the cache security groups associated with the cache cluster. • One or more EC2 security groups have been revoked from any of the cache security groups associated with the cache 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. **Note** The DNS entry for the replaced cache node is not changed. 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. **Note** The DNS entry for the replaced cache node is not changed. 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. Message (Memcached): `"Cache node %s shutdown"` Then a second message: `"Cache node %s restarted"`

Event Name	Message	Description
ElastiCache:CertificateRenewalComplete (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 cache cluster and all associated cache nodes has completed.
ElastiCache:FailoverComplete (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. For information on what actions you can take, see Replacing nodes.
ElastiCache:NodeReplacementScheduled	`ElastiCache:NodeReplacementScheduled : cluster-name`	A node in your cluster is scheduled for replacement during the window described in the notification. For information on what actions you can take, see Replacing nodes.
ElastiCache:RemoveCacheNodeComplete	`ElastiCache:RemoveCacheNodeComplete : cluster-name`	A cache node has been removed from the cache 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.
ElastiCache:ServiceUpdateAvailableForNode	`"Service update is available for cache node %s."`	A self-service update is available for the node.

Event Name	Message	Description
ElastiCache:SnapshotComplete (Redis OSS only)	`ElastiCache:SnapshotComplete : ` *`cluster-name`*	A cache snapshot has completed successfully.
ElastiCache:SnapshotFailed (Redis OSS only)	`SnapshotFailed : ` *`cluster-name`*	A cache snapshot has failed. See the cluster's cache events for more a detailed cause. If you describe the snapshot, see `DescribeSnapshots` , the status will be `failed`.

Log delivery

> **Note**
>
> Redis OSS Slow Log is supported for Redis OSS cache clusters and replication groups using engine version 6.0 onward.
>
> Redis OSS Engine Log is supported for Redis OSS cache clusters and replication groups using engine version 6.2 onward.

Log delivery lets you stream Redis OSS SLOWLOG or **Redis OSS 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 Redis OSS 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 ElastiCache (Redis OSS) Slow Log contains the following information:

- **CacheClusterId –** The ID of the cache 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 (Redis OSS) 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

The ElastiCache (Redis OSS) Engine Log contains the following information:

- **CacheClusterId –** The ID of the cache 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 –** Redis OSS 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 10000 milliseconds.
```

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

- You specify a CloudWatch Logs log group where the logs will be delivered.
- Logs from multiple 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 cache 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 (Redis OSS) to send logs to a CloudWatch Logs log group:

```
{
    "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 Redis OSS clusters and replication groups can be delivered to the same delivery stream.
- Logs from each node within a cache 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 (Redis OSS) to send logs to an Amazon Kinesis Data Firehose delivery stream.

```
{
    "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": "Amazon Kinesis Data Firehose delivery stream ARN",
            "Effect": "Allow"
        }
    ]
}
```

Specifying log delivery using the Console

Using the AWS Management Console you can create a Redis OSS (cluster mode disabled) cluster by following the steps at Creating a Redis OSS (cluster mode disabled) cluster (Console) or create a Redis OSS (cluster mode enabled) cluster using the steps at Creating a Redis OSS (cluster mode enabled) cluster (Console). In either case, you configure log delivery by doing the following;

1. Under **Advanced Redis OSS 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 **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 cache cluster's performance starts to degrade. For more information, see Which Metrics Should I Monitor? in this guide.

Topics

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 Redis OSS

Metric	Description	Unit
`CPUUtilization`	The percentage of CPU utilization for the entire host. Because Redis OSS is single-	Percent

Metric	Description	Unit
	threaded, we recommend you monitor `EngineCPUUtilization` metric for nodes with 4 or more vCPUs.	
`CPUCreditBalance`	The number of earned CPU credits that an instance has accrued since it was launched or started. For T2 Standard, the CPUCredit Balance also includes the number of launch credits that have been accrued. 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 . For T2 Standard, launch credits do not count towards the limit. The credits in the CPUCreditBalance are available for the instance to spend to burst beyond its baseline CPU utilization. CPU credit metrics are available at a five-minu te frequency only. This metrics is not available for T2 burstable performance instances.	Credits (vCPU-min utes)

Metric	Description	Unit
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). 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. This metrics is not available for T2 burstable performance instances.	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
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

Metric	Description	Unit
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
NetworkBandwidthInAllowanceExceeded	The number of packets queued or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance.	Count
NetworkConntrackAllowanceExceeded	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
NetworkBandwidthOutAllowanceExceeded	The number of packets queued or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance.	Count
NetworkPacketsPerSecondAllowanceExceeded	The number of packets queued or dropped because the bidirectional packets per second exceeded the maximum for the instance.	Count
NetworkMaxBytesIn	The maximum burst of received bytes within each minute.	Bytes
NetworkMaxBytesOut	The maximum burst of transmitted bytes within each minute.	Bytes
NetworkMaxPacketsIn	The maximum burst of received packets within each minute.	Count
NetworkMaxPacketsOut	The maximum burst of transmitted packets within each minute.	Count
SwapUsage	The amount of swap used on the host.	Bytes

Metrics for Redis OSS

The `AWS/ElastiCache` namespace includes the following Redis OSS metrics.

With the exception of `ReplicationLag` and `EngineCPUUtilization`, these metrics are derived from the Redis OSS **info** command. Each metric is calculated at the cache node level.

For complete documentation of the Redis OSS **info** command, see http://redis.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 Redis OSS INFO.	Number
`AuthenticationFailures`	The total number of failed attempts to authenticate to 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
`BytesUsedForCache`	The total number of bytes allocated by Redis OSS for all purposes, including the dataset, buffers, and so on.	Bytes
	`Dimension: Tier=Memory` for Redis OSS clusters using Data tiering: The total number of bytes used for cache by memory. This is the value of `used_memory` statistic at Redis OSS INFO.	Bytes

Metric	Description	Unit
	`Dimension: Tier=SSD` for Redis OSS clusters using Data tiering: 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.	Bytes
`BytesWrittenToDisk`	The total number of bytes written to disk per minute. Supported only for clusters using Data tiering.	Bytes
`CacheHits`	The number of successful read-only key lookups in the main dictionary. This is derived from `keyspace_hits` statistic at Redis OSS INFO.	Count
`CacheMisses`	The number of unsuccessful read-only key lookups in the main dictionary. This is derived from `keyspace_misses` statistic at Redis OSS 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 Redis OSS instance. If the cache ratio is lower than about 0.8, 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: `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 two to four of the connections to monitor the cluster in each case. This is derived from the `connected_clients` statistic at [Redis OSS INFO](#).	Count
`CurrItems`	The number of items in the cache. This is derived from the Redis OSS `keyspace` statistic, summing all of the keys in the entire keyspace.	Count
	`Dimension: Tier=Memory` for clusters using [Data tiering](#). The number of items in memory.	Count
	`Dimension: Tier=SSD` (solid state drives) for clusters using [Data tiering](#). 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 Redis OSS expires statistic, summing all of the keys with a ttl set in the entire keyspace.	Count
DatabaseCapacityUsagePercentage	Percentage of the total data capacity for the cluster that is in use. 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 Redis OSS INFO. In all other cases, the metric is calculated using used_memory/maxmemory .	Percent
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: used_memory - mem_not_counted_for_evict/maxmemory On Data Tiered instances, the metric is calculated as: (used_memory + SSD used) / (maxmemory + SSD total capacity) where used_memory and maxmemory are taken from Redis OSS INFO	Percent

Metric	Description	Unit
DatabaseMemoryUsagePercentage	Percentage of the memory for the cluster that is in use. This is calculated using used_memory/maxmemory from Redis OSS 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 Redis OSS INFO.	Percent
DB0AverageTTL	Exposes avg_ttl of DBO from the keyspace statistic of Redis OSS INFO command. Replicas don't expire keys, instead they wait for primary nodes to expire keys. When a primary node expires a key (or evicts it because of LRU), it synthesizes a DEL command, which is transmitted to all the replicas. Therefore, DB0AverageTTL is 0 for replica nodes, due the fact that they don't expire keys, and thus don't track TTL.	Milliseconds

Metric	Description	Unit
`EngineCPUUtilization`	Provides CPU utilization of the Redis OSS engine thread. Because Redis OSS is single-threaded, you can use this metric to analyze the load of the Redis OSS process itself. The `EngineCPUUtilization` metric provides a more precise visibility of the Redis OSS 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. **Note** 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 Redis OSS and high CPU usage from the background monitoring processes. Therefore, we recommend monitorin	Percent

Metric	Description	Unit
	g the `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 Redis OSS 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 Redis OSS, the lag indicates the maximum delay among the shards.	Seconds
`IamAuthenticationExpirations`	The total number of expired IAM-authenticated Redis OSS connections. You can find more information about Authenticating with IAM in the user guide.	Count
`IamAuthenticationThrottling`	The total number of throttled IAM-authenticated Redis OSS AUTH or HELLO requests. You can find more information about Authenticating with IAM 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 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 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 Redis OSS 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 Redis OSS INFO. **Note** If you are using ElastiCache (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 (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.	Count
NumItemsWrittenToDisk	The total number of items written to disk per minute. Supported only for clusters using Data tiering.	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 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

Metric	Description	Unit
`Reclaimed`	The total number of key expiration events. This is derived from the `expired_keys` statistic at Redis OSS INFO.	Count
`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 Redis OSS 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 Redis OSS engine version 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 backgroun d save process. This is derived from the `rdb_bgsave_in_progress` statistic at Redis OSS INFO.	Boolean

Metric	Description	Unit
TrafficManagementActive	Indicates whether ElastiCache (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 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. **Note** 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

EngineCPUUtilization availability

AWS Regions listed following are available on all supported node types.

Region	Region name
us-east-2	US East (Ohio)
us-east-1	US East (N. Virginia)
us-west-1	US West (N. California)
us-west-2	US West (Oregon)

Region	Region name
`ap-northeast-1`	Asia Pacific (Tokyo)
`ap-northeast-2`	Asia Pacific (Seoul)
`ap-northeast-3`	Asia Pacific (Osaka)
`ap-east-1`	Asia Pacific (Hong Kong)
`ap-south-1`	Asia Pacific (Mumbai)
`ap-southeast-1`	Asia Pacific (Singapore)
`ap-southeast-2`	Asia Pacific (Sydney)
`ap-southeast-3`	Asia Pacific (Jakarta)
`ca-central-1`	Canada (Central)
`cn-north-1`	China (Beijing)
`cn-northwest-2`	China (Ningxia)
`me-south-1`	Middle East (Bahrain)
`eu-central-1`	Europe (Frankfurt)
`eu-west-1`	Europe (Ireland)
`eu-west-2`	Europe (London)
`eu-west-3`	EU (Paris)
`eu-south-1`	Europe (Milan)
`af-south-1`	Africa (Cape Town)
`eu-north-1`	Europe (Stockholm)
`sa-east-1`	South America (São Paulo)

Region	Region name
`us-gov-west-1`	AWS GovCloud (US-West)
`us-gov-east-1`	AWS GovCloud (US-East)

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 [Redis OSS 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 [redis commands]() in the Redis OSS documentation.

Metric	Description	Unit
`ClusterBasedCmds`	The total number of commands that are cluster-based. This is derived from the Redis OSS `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 Redis OSS `commandstats` statistic by summing **eval**, **evalsha**.	Count
`EvalBasedCmdsLatency`	Latency of eval-based commands.	Microseconds

Metric	Description	Unit
`GeoSpatialBasedCmds`	The total number of commands for geospatial-based commands. This is derived from the Redis OSS `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
`GetTypeCmds`	The total number of **read-only** type commands. This is derived from the Redis OSS `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 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
`HashBasedCmdsLatency`	Latency of hash-based commands.	Microseconds
`HyperLogLogBasedCmds`	The total number of `HyperLogLog` -based commands. This is derived from the Redis OSS `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

Metric	Description	Unit
JsonBasedCmds	The total number of JSON commands, including both read and write commands. This is derived from the Redis OSS 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
JsonBasedGetCmds	The total number of JSON read-only commands. This is derived from the Redis OSS 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 Redis OSS 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 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
KeyBasedCmdsLatency	Latency of key-based commands.	Microseconds

Metric	Description	Unit
`ListBasedCmds`	The total number of commands that are list-based. This is derived from the 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
`ListBasedCmdsLatency`	Latency of list-based commands.	Microseconds
NonKeyTypeCmds	The total number of commands that are not key-based. This is derived from the Redis OSS `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 Redis OSS `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 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
`SetBasedCmdsLatency`	Latency of set-based commands.	Microseconds

Metric	Description	Unit
`SetTypeCmds`	The total number of **write** types of commands. This is derived from the 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
`SetTypeCmdsLatency`	Latency of write commands.	Microseconds
`SortedSetBasedCmds`	The total number of commands that are sorted set-based. This is derived from the 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
`SortedSetBasedCmdsLatency`	Latency of sorted-based commands.	Microseconds
`StringBasedCmds`	The total number of commands that are string-based. This is derived from the Redis OSS `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 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

Metric	Description	Unit
StreamBasedCmdsLatency	Latency of stream-based commands.	Microseconds

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
- Evictions
- CurrConnections
- Memory
- Network
- Latency
- Replication
- Traffic Management

CPUUtilization

This is a host-level metric reported as a percentage. For more information, see Host-Level Metrics.

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 Redis OSS is 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 cache cluster out by adding read replicas. If the main workload is from write requests, depending on your cluster configuration, we recommend that you:

- **Redis OSS (cluster mode disabled) clusters:** scale up by using a larger cache instance type.

- **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`, Redis OSS users might be able to use the Redis OSS metric `EngineCPUUtilization`, 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 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 Redis OSS engine core. To see if this metric is available on your nodes and for more information, see Metrics for Redis OSS.

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 Redis OSS.

For more information, see the **CPUs** section at Monitoring best practices with Amazon ElastiCache (Redis OSS) using Amazon CloudWatch.

SwapUsage

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 that you have enough memory to create a Redis OSS snapshot
- Managing Reserved Memory

Evictions

This is a cache engine metric. We recommend that you determine your own alarm threshold for this metric based on your application needs.

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 (Redis OSS) using Amazon CloudWatch.

Memory

Memory is a core aspect of 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 Redis OSS INFO command.

For more information, see the **Memory** section at Monitoring best practices with Amazon ElastiCache (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 (Redis OSS) using Amazon CloudWatch.

Latency

You can measure a command's latency with a set of CloudWatch metrics that provide aggregated latencies per data structure. These latency metrics are calculated using the `commandstats` statistic from the Redis OSS INFO command.

For more information, see the **Latency** section at Monitoring best practices with Amazon ElastiCache (Redis OSS) 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 (Redis OSS) using Amazon CloudWatch].

Traffic Management

ElastiCache (Redis OSS) automatically manages traffic against a node when more incoming commands are sent to the node than can be processed by 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 examples require the CloudWatch command line tools. For more information about CloudWatch and to download the developer tools, see the CloudWatch product page.

The following procedures show you how to use CloudWatch to gather storage space statistics for an cache cluster for the past hour.

> **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 gather CPU utilization statistics for a cache cluster

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

2. Select the cache nodes you want to view metrics for.

> **Note**
>
> Selecting more than 20 nodes disables viewing metrics on the console.

   a. On the **Cache Clusters** page of the AWS Management Console, click the name of one or more cache clusters.

      The detail page for the cache cluster appears.

   b. Click the **Nodes** tab at the top of the window.

   c. On the **Nodes** tab of the detail window, select the cache nodes that you want to view metrics for.

      A list of available CloudWatch Metrics appears at the bottom of the console window.

   d. Click on the **CPU Utilization** metric.

      The CloudWatch console will open, displaying your selected metrics. You can use the **Statistic** and **Period** drop-down list boxes and **Time Range** tab to change the metrics being displayed.

Monitoring CloudWatch Cluster and Node Metrics using the CloudWatch 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=2013-07-05T00:00:00`
  - `EndTime=2013-07-06T00:00:00`
  - `Period=60`
  - `MeasureName=CPUUtilization`
  - `Dimensions=CacheClusterId=mycachecluster,CacheNodeId=0002`

  Example

```
http://monitoring.amazonaws.com/
    ?Action=GetMetricStatistics
    &SignatureVersion=4
    &Version=2014-12-01
    &StartTime=2018-07-05T00:00:00
    &EndTime=2018-07-06T23:59:00
    &Period=3600
    &Statistics.member.1=Average
```

```
&Dimensions.member.1="CacheClusterId=mycachecluster"
&Dimensions.member.2="CacheNodeId=0002"
&Namespace=&AWS;/ElastiCache
&MeasureName=CPUUtilization
&Timestamp=2018-07-07T17%3A48%3A21.746Z
&AWS;AccessKeyId=<&AWS; Access Key ID>
&Signature=<Signature>
```

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.

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. 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 Redis OSS, in the navigation pane choose **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
```

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>
```

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 cache 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 cache 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 cache 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 cache 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 cache 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 cache cluster on which notifications are enabled.
- The Amazon SNS topic you are publishing to cannot be encrypted.

Note

It is possible to attach an encrypted (at-rest) Amazon SNS topic to the cluster. However, the status of the topic from the ElastiCache console will show as inactive, which effectively disassociates the topic from the cluster when ElastiCache pushes messages to the topic.

- The Amazon SNS topic has to be in the same Region as the ElastiCache cluster.

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 cache 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 cache cluster parameters have been changed.
ElastiCache:CacheClusterProvisioningComplete	`ElastiCache:CacheClusterProvisioningComplete cluster-name-0001-005`	The provisioning of a cache cluster is completed, and the cache nodes in the cache cluster are ready to use.
ElastiCache:CacheClusterProvisioningFailed due to incompatible network state	`ElastiCache:CacheClusterProvisioning`	An attempt was made to launch a new cache cluster

Event Name	Message	Description
	Failed : *cluster-name*	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: • The list of cache security groups authorized for the cache cluster has been modified. • One or more new EC2 security groups have been authorized on any of the cache security groups associated with the cache cluster. • One or more EC2 security groups have been revoked from any of the cache security groups associated with the cache 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. **Note** The DNS entry for the replaced cache node is not changed. 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. **Note** The DNS entry for the replaced cache node is not changed. 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. Message (Memcached): `"Cache node %s shutdown"` Then a second message: `"Cache node %s restarted"`

Event Name	Message	Description
ElastiCache:CertificateRenewalComplete (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 cache cluster and all associated cache nodes has completed.
ElastiCache:FailoverComplete (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. For information on what actions you can take, see Replacing nodes.
ElastiCache:NodeReplacementScheduled	`ElastiCache:NodeReplacementScheduled : cluster-name`	A node in your cluster is scheduled for replacement during the window described in the notification. For information on what actions you can take, see Replacing nodes.
ElastiCache:RemoveCacheNodeComplete	`ElastiCache:RemoveCacheNodeComplete : cluster-name`	A cache node has been removed from the cache 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.
ElastiCache:ServiceUpdateAvailableForNode	`"Service update is available for cache node %s."`	A self-service update is available for the node.

Event Name	Message	Description
ElastiCache:SnapshotComplete (Redis OSS only)	`ElastiCache:SnapshotComplete : cluster-name`	A cache snapshot has completed successfully.
ElastiCache:SnapshotFailed (Redis OSS only)	`SnapshotFailed : cluster-name`	A cache snapshot has failed. See the cluster's cache events for more a detailed cause. If you describe the snapshot, see `DescribeSnapshots`, the status will be `failed`.

Related topics

- Viewing ElastiCache events

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.m1.small",

    },
    "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.m1.small",

        "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.m1.small",
        "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"
}
```

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.

Resource	Default
Serverless caches per region	40
Serverless snapshots per day per cache	24
Nodes per Region	300
Nodes per cluster per instance type (Redis OSS cluster mode enabled)	90
Nodes per shard (Redis OSS cluster mode disabled)	6
Parameter groups per Region	300
Security groups per Region	50
Subnet groups per Region	300
Subnets per subnet group	20
Users per user group	100
Maximum number of users	1000

Resource	Default
Maximum number of user groups	100

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

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 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 cache cluster capacity

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 (Redis OSS) .

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 (Redis OSS) specific notifications

There are currently no outstanding ElastiCache (Redis OSS) notifications.

ElastiCache (Redis OSS) Documentation history

- **API version:** 2015-02-02
- **Latest documentation update:** November 27, 2023

The following table describes important changes in each release of the *ElastiCache (Redis OSS) User Guide* after March 2018. For notification about updates to this documentation, you can subscribe to the RSS feed.

Recent ElastiCache (Redis OSS) Updates

Change	Description	Date
ElastiCache (Redis OSS) added support for additional C7gn node sizes	ElastiCache (Redis OSS) added support for additional C7gn node sizes.	January 10, 2024
ElastiCache (Redis OSS) now supports creation of serverles s caches	You can now create serverles s caches, which simplify cache management and instantly scale to support the most demanding applications. For more information, see Choosing between deploymen t options. As part of this feature, new permissions were added to `ElastiCac heServiceRolePolicy` and `AmazonElastiCacheF ullAccess` to allow association of serverless caches with managed VPC endpoints. Additionally, permissions were added to support a revised console experience using `AmazonEla`	November 27, 2023

	`stiCacheFullAccess` policy.	
ElastiCache (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 (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 (Redis OSS) now supports authenticating users using IAM	IAM Authentication allows you to authenticate a connection to ElastiCache (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 (Redis OSS) now supports Redis OSS 7	This release brings several new features to Amazon ElastiCache (Redis OSS): Redis OSS functions, ACL improvements and Sharded Pub/Sub. For more informati on, see ElastiCache (Redis OSS) version 7.0.	November 8, 2022
ElastiCache (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 (Redis OSS) now supports native JavaScript 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 JavaScript 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 (Redis OSS) now supports Redis OSS 6.2 and Data Tiering	Amazon ElastiCache (Redis OSS) introduces the next version of the Redis OSS engine supported by Amazon ElastiCache. ElastiCache (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 (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 (Redis OSS) version 6.2 (enhanced) and Data tiering.	November 23, 2021

Support for Auto Scaling	ElastiCache (Redis OSS) now supports Auto Scaling. ElastiCache (Redis OSS) auto scaling is the ability to increase or decrease the desired shards or replicas in your ElastiCache (Redis OSS) service automatically. ElastiCache leverages the Application Auto Scaling service to provide this functionality. For more information, see Auto Scaling ElastiCache (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
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 for Redis OSS or Using Outposts for Memcached.	October 8, 2020

ElastiCache now supports Redis OSS 6	Amazon ElastiCache (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 (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 geographically 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 (Redis OSS) now supports scaling your Redis OSS Cluster environment 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 ElastiCache (Redis OSS) version 5.0.6 onwards in all AWS Regions and for all existing and new ElastiCache (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 permissions by specifying ElastiCache resources in an AWS Identity and Access Management (IAM) policy. For more information, see Resource-level permissions.	August 12, 2020
ElastiCache (Redis OSS) adds additional Amazon CloudWatch metrics	ElastiCache (Redis OSS) now supports new CloudWatch metrics, including `PubSubCmds` and `HyperLogLogBasedCmds`. For a full list, see Metrics for Redis.	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 (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 (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 (Redis OSS) now supports Redis OSS version 5.0.6	For more information, see ElastiCache (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 (Redis OSS) server	ElastiCache (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 AWS 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 (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. ElastiCac he (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 (Redis OSS) now allows users to use a single reader endpoint for your Amazon ElastiCache (Redis OSS) cluster.	This feature allows you to direct all read traffic to your ElastiCache (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 (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 (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 (Redis OSS) cluster. For more information, see Shards.	November 19, 2018

ElastiCache (Redis OSS) support for autofailover and backup and restore on all T2 nodes	ElastiCache (Redis OSS) introduces support for autofailover, creating snapshots, and backup and restore on all T2 nodes. For more information, see ElastiCache (Redis OSS) Backup and Restore and Snapshot.	November 19, 2018
ElastiCache (Redis OSS) support for M5 and R5 nodes	ElastiCache (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 (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 (Redis OSS) User Guide*. See also DecreaseReplicaCount and IncreaseReplicaCount in the *ElastiCache API Reference.*	September 17, 2018

FedRAMP compliance certification	ElastiCache (Redis OSS) is now certified for FedRAMP compliance. For more information, see Compliance validation for Amazon ElastiCache.	August 30, 2018
Redis OSS (cluster mode enabled) engine upgrades	Amazon ElastiCache (Redis OSS) has added support for upgrading Redis OSS (cluster mode enabled) engine versions. For more information, see Upgrading Engine Versions.	August 20, 2018
PCI DSS compliance certification	ElastiCache (Redis OSS) is now certified for PCI DSS compliance. For more information, see Compliance validation for Amazon ElastiCache.	July 5, 2018
Support for ElastiCache (Redis OSS) 4.0.10	ElastiCache (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 (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 (Redis OSS) User Guide) and for Memcached (ElastiCache (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 (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 (Redis OSS) 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: • 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: • Supported Regions • 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: • Supported Regions • 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: • Using Service-Linked Roles for Amazon ElastiCache • 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: • Supported cache node types • Redis OSS node-type specific parameters	November 20, 2017
ElastiCache (Redis OSS) 3.2.10 and support for online resharding	Amazon ElastiCache (Redis OSS) adds support for ElastiCache (Redis OSS) 3.2.10. ElastiCache (Redis OSS) also introduces online cluster resizing to add or remove shards from the cluster while it continues to serve incoming I/O requests. For more information, see the following: • Online cluster resizing • Online resharding and shard rebalancing for Redis OSS (cluster mode enabled)	November 9, 2017
HIPAA eligibility	ElastiCache (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: • Compliance validation for Amazon ElastiCache • Data security in Amazon ElastiCache	November 2, 2017

Change	Description	Date Changed
ElastiCache (Redis OSS) 3.2.6 and support for encryption	ElastiCache adds support for ElastiCache (Redis OSS) 3.2.6, which includes two encryption features: • 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. • At-rest encryption encrypts your on-disk data during sync and backup operations. For more information, see the following: • Data security in Amazon ElastiCache • Supported ElastiCache (Redis OSS) versions	October 25, 2017
Connection patterns topic	ElastiCache documentation adds a topic covering various patterns for accessing an ElastiCache cluster in an Amazon VPC. 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 testing Automatic Failover	ElastiCache adds support for testing Automatic Failover on Redis OSS clusters that support rep lication. For more information, see the following: • Testing automatic failover in the *ElastiCache User Guide*. • TestFailover in the *ElastiCache API Reference*. • test-failover in the *AWS CLI Reference*.	April 4, 2017

Change	Description	Date Changed
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 ElastiCache for Redis OSS. For more information, see the following: • Managing Reserved Memory • New parameters for Redis OSS 3.2.4	March 15, 2017
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: • Supported Regions • Supported cache node types	December 13, 2016

Change	Description	Date Changed
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: • Supported Regions • 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: • Supported Regions • 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: • Supported Regions • 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. This release includes a redesigned managemen t console that uses terminology in keeping with industry usage. For more information, see the following: • Comparing Memcached and Redis OSS • ElastiCache components and features — note the sections on Nodes, Shards, Clusters, and Replication. • ElastiCache (Redis OSS) 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: • Supported cache node types • 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: • Supported cache node types • 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 ElastiCache. For more information, see the following: • Exporting a backup in the *Amazon ElastiCache User Guide* • 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 (Redis OSS) .	March 24, 2016
Easy engine upgrade	ElastiCache adds the ability to easily upgrade your Redis OSS cache engine. For more information, see Engine versions and upgrading .	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 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 addresses. For more information, see the following: • ElastiCache (Redis OSS) version 2.8.24 (enhanced) • 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 information, see Supported ElastiCache (Redis OSS) versions.	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, disconnects clients who attempt to write to a read-only replica. For more information, see ElastiCache (Redis OSS) version 2.8.23 (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 enhancements and improvements since version 2.8.21. Improvements include: • Implementation of a forkless save process that enables a successful save when low available memory could cause a forked save to fail. • Additional CloudWatch metrics — *SaveInProgress* and *ReplicationBytes*. • To enable partial synchronizations, the Redis OSS parameter `repl-backlog-size` now applies to all clusters. For a complete list of changes and more information, see [ElastiCache (Redis OSS) version 2.8.22 (enhanced)](). This documentation release includes a reorganization 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
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

Change	Description	Date Changed
New topic: Accessing ElastiCache from outside AWS	Added new topic on how to access ElastiCache resources from outside AWS. For more information, see Accessing ElastiCache from outside AWS.	July 9, 2015
Node replacement messages added	ElastiCache adds three messages pertaining to scheduled node replacement, ElastiCache:NodeReplacementScheduled, ElastiCache:NodeReplacementRescheduled, and ElastiCache:NodeReplacementCanceled. 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

Change	Description	Date Changed
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: • The HyperLogLog data structure, with the Redis OSS commands PFADD, PFCOUNT, and PFMERGE. • Lexicographic range queries with the new commands ZRANGEBYLEX, ZLEXCOUNT, and ZREMRANGEBYLEX. • 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. For more information on HyperLogLog, see [Redis OSS new data structure: the HyperLogLog](#). For more information on PFADD, PFCOUNT, and PFMERGE, see the [Redis OSS Documentation](#) and click **HyperLogLog**.	March 11, 2015
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

Change	Description	Date Changed
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 (Redis OSS) with Multi-AZ.	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 parameter groups.	September 11, 2014
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 parameter groups.	July 1, 2014

Change	Description	Date Changed
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 cache cluster with their existing data from a Redis OSS snapshot file, easing migration to a managed ElastiCache environment. 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 automatically 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, cache clusters are created in an Amazon VPC by default. For more information, see Amazon VPCs and ElastiCache security.	January 8, 2013
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' cache 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

Change	Description	Date Changed
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.	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*.