

# Distributed Cache Service (DCS)

## User Guide

**Issue** 01  
**Date** 2024-11-25



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **Huawei Cloud Computing Technologies Co., Ltd.**

Address: Huawei Cloud Data Center Jiaoxinggong Road  
Qianzhong Avenue  
Gui'an New District  
Gui Zhou 550029  
People's Republic of China

Website: <https://www.huaweicloud.com/intl/en-us/>

---

# Contents

---

<b>1 Service Overview</b>	<b>1</b>
1.1 What Is DCS?	1
1.2 Application Scenarios	3
1.3 DCS Instance Types	3
1.3.1 Single-Node Redis	3
1.3.2 Master/Standby Redis	5
1.3.3 Proxy Cluster Redis	9
1.3.4 Redis Cluster	13
1.4 DCS Instance Specifications	15
1.4.1 Redis 3.0 Instance Specifications (Obsolete)	15
1.4.2 Redis 4.0 and 5.0 Instance Specifications	18
1.4.3 Redis 6.0 Instance Specifications	30
1.5 Command Compatibility	38
1.5.1 Commands Supported and Disabled by DCS for Redis 3.0	39
1.5.2 Commands Supported and Disabled by DCS for Redis 4.0	42
1.5.3 Commands Supported and Disabled by DCS for Redis 5.0	49
1.5.4 Commands Supported and Disabled by DCS for Redis 6.0	56
1.5.5 Web CLI Commands	60
1.5.6 Command Restrictions	63
1.5.7 Other Command Usage Restrictions	69
1.6 HA	71
1.7 Comparing Redis Versions	72
1.8 Comparing DCS and Open-Source Cache Services	74
1.9 Basic Concepts	76
1.10 Permissions Management	77
1.11 Related Services	84
<b>2 Getting Started</b>	<b>86</b>
2.1 Creating an Instance	86
2.1.1 Identifying Requirements	86
2.1.2 Preparing Required Resources	86
2.1.3 Creating a DCS Redis Instance	88
2.2 Accessing an Instance	90
2.2.1 Network Conditions for Accessing DCS Redis	90

2.2.2 Accessing a DCS Redis Instance Through redis-cli.....	91
2.2.3 Access in Different Languages.....	94
2.2.3.1 Java.....	94
2.2.3.1.1 Connecting to Redis on Jedis (Java).....	94
2.2.3.1.2 Connecting to Redis on Lettuce (Java).....	101
2.2.3.1.3 Connecting to Redis on Redisson (Java).....	114
2.2.3.2 Connecting to Redis on redis-py (Python).....	123
2.2.3.3 Connecting to Redis on go-redis (Go).....	126
2.2.3.4 Connecting to Redis on hiredis (C++).....	127
2.2.3.5 Connecting to Redis on StackExchange.Redis (C#).....	130
2.2.3.6 PHP.....	132
2.2.3.6.1 Connecting to Redis on phpredis (PHP).....	132
2.2.3.6.2 Connecting to Redis on predis (PHP).....	134
2.2.3.7 Connecting to Redis on ioredis (Node.js).....	135
2.2.4 Accessing a DCS Redis Instance on the Console.....	138
2.3 Viewing Details of a DCS Instance.....	139
<b>3 User Guide.....</b>	<b>142</b>
3.1 Permissions Management.....	142
3.1.1 Creating a User and Granting DCS Permissions.....	142
3.1.2 DCS Custom Policies.....	143
3.2 Operating DCS Instances.....	144
3.2.1 Modifying DCS Instance Specifications.....	144
3.2.2 Restarting DCS Instances.....	148
3.2.3 Deleting DCS Instances.....	149
3.2.4 Performing a Master/Standby Switchover for a DCS Instance.....	151
3.2.5 Clearing DCS Instance Data.....	151
3.2.6 Exporting DCS Instance List.....	152
3.3 Managing DCS Instances.....	152
3.3.1 Configuration Notice.....	152
3.3.2 Modifying Configuration Parameters.....	153
3.3.3 Modifying Maintenance Time Window.....	163
3.3.4 Modifying the Security Group.....	164
3.3.5 Viewing Background Tasks.....	165
3.3.6 Viewing Data Storage Statistics of a DCS Redis 3.0 Proxy Cluster Instance.....	165
3.3.7 Managing Shards and Replicas.....	166
3.3.8 Analyzing Big Keys and Hot Keys.....	167
3.3.9 Scanning and Deleting Expired Keys in a DCS Redis Instance.....	169
3.3.10 Managing IP Address Whitelist.....	173
3.3.11 Viewing Redis Run Logs.....	174
3.3.12 Diagnosing an Instance.....	175
3.4 Backing Up and Restoring DCS Instances.....	175
3.4.1 Overview.....	176

3.4.2 Configuring an Automatic Backup Policy.....	178
3.4.3 Manually Backing Up a DCS Instance.....	179
3.4.4 Restoring a DCS Instance.....	180
3.4.5 Downloading a Backup File.....	181
3.5 Migrating Data with DCS.....	182
3.5.1 Introduction to Migration with DCS.....	182
3.5.2 Importing Backup Files.....	184
3.5.2.1 Importing Backup Files from an OBS Bucket.....	184
3.5.2.2 Importing Backup Files from Redis.....	187
3.5.3 Migrating Data Online.....	188
3.6 Managing Passwords.....	193
3.6.1 DCS Instance Passwords.....	193
3.6.2 Changing Instance Passwords.....	193
3.6.3 Resetting Instance Passwords.....	194
3.6.4 Changing Password Settings for DCS Redis Instances.....	195
3.7 Monitoring.....	196
3.7.1 DCS Metrics.....	196
3.7.2 Viewing DCS Monitoring Metrics.....	225
<b>4 Best Practices.....</b>	<b>227</b>
4.1 Service Application.....	227
4.1.1 Serializing Access to Frequently Accessed Resources.....	227
4.1.2 Ranking with DCS.....	232
4.2 Network Connection.....	235
4.2.1 Configuring Redis Client Retry.....	235
4.3 Usage Guide.....	241
4.3.1 Suggestions on Using DCS.....	241
<b>5 FAQs.....</b>	<b>251</b>
5.1 Instance Types/Versions.....	251
5.1.1 Comparing Versions.....	251
5.1.2 How Do I View the Version of a DCS Redis Instance?.....	253
5.1.3 New Features of DCS for Redis 4.0.....	253
5.1.4 New Features of DCS for Redis 5.0.....	257
5.1.5 New Features of DCS for Redis 6.0.....	264
5.2 Client and Network Connection.....	266
5.2.1 Security Group Configurations.....	266
5.2.2 Does DCS Support Access at EIPs?.....	267
5.2.3 Does DCS Support Cross-VPC Access?.....	267
5.2.4 What Should I Do If Access to DCS Fails After Server Disconnects?.....	268
5.2.5 Why Do Requests Sometimes Time Out in Clients?.....	268
5.2.6 What Should I Do If an Error Is Returned When I Use the Jedis Connection Pool?.....	268
5.2.7 Why Is "ERR unknown command" Displayed When I Access a DCS Redis Instance Through a Redis Client?.....	270

5.2.8 How Do I Access a DCS Redis Instance Through Redis Desktop Manager?.....	271
5.2.9 What If "ERR Unsupported CONFIG subcommand" is Displayed in SpringCloud?.....	272
5.2.10 How Do I Troubleshoot Redis Connection Failures?.....	272
5.2.11 What Should Be Noted When Using Redis for Pub/Sub?.....	273
5.2.12 Should I Use a Domain Name or an IP Address to Connect to a DCS Redis Instance?.....	274
5.3 Redis Usage.....	274
5.3.1 Why Is CPU Usage of a DCS Redis Instance 100%?.....	274
5.3.2 Can I Change the VPC and Subnet for a DCS Redis Instance?.....	274
5.3.3 Why Aren't Security Groups Configured for DCS Redis 4.0 and Later Instances?.....	275
5.3.4 Do DCS Redis Instances Limit the Size of a Key or Value?.....	275
5.3.5 Can I Obtain the Addresses of the Nodes in a Cluster DCS Redis Instance?.....	275
5.3.6 Why Is Available Memory Smaller Than Instance Cache Size?.....	275
5.3.7 Does DCS for Redis Support Multiple Databases?.....	276
5.3.8 Does DCS for Redis Support Redis Clusters?.....	276
5.3.9 Does DCS for Redis Support Sentinel?.....	276
5.3.10 What Is the Default Data Eviction Policy?.....	276
5.3.11 What Should I Do If an Error Occurs in Redis Exporter?.....	277
5.3.12 Why Is Redisson Distributed Lock Not Supported by DCS Proxy Cluster Redis 3.0 Instances?.....	277
5.3.13 Can I Customize or Change the Port for Accessing a DCS Instance?.....	277
5.3.14 Can I Modify the Connection Addresses for Accessing a DCS Instance?.....	278
5.3.15 Does DCS Support Cross-AZ Deployment?.....	278
5.3.16 Why Does It Take a Long Time to Start a Cluster DCS Instance?.....	278
5.3.17 What If Redis Commands Are Incompatible with DCS for Redis?.....	278
5.3.18 Does DCS for Redis Provide Backend Management Software?.....	279
5.3.19 Why Is Memory of a DCS Redis Instance Used Up by Just a Few Keys?.....	279
5.3.20 Can I Recover Data from Deleted DCS Instances?.....	279
5.3.21 Why Is "Error in execution" Returned When I Access Redis?.....	279
5.4 Redis Commands.....	280
5.4.1 How Do I Clear Redis Data?.....	280
5.4.2 How Do I Disable High-Risk Commands?.....	280
5.4.3 Does DCS for Redis Support Pipelining?.....	281
5.4.4 Does DCS for Redis Support the INCR and EXPIRE Commands?.....	281
5.4.5 Why Do I Fail to Execute Some Redis Commands?.....	281
5.4.6 Why Does a Redis Command Fail to Take Effect?.....	282
5.4.7 Is There a Time Limit on Executing Redis Commands? What Will Happen If a Command Times Out? .....	282
5.5 Instance Scaling and Upgrade.....	283
5.5.1 Can DCS Redis Instances Be Upgraded, for Example, from Redis 4.0 to 5.0?.....	283
5.5.2 Are Services Interrupted If Maintenance is Performed During the Maintenance Time Window?.....	283
5.5.3 Are Instances Stopped or Restarted During Specification Modification?.....	283
5.5.4 Are Services Interrupted During Specification Modification?.....	283
5.5.5 Why Can't I Modify Specifications for a DCS Redis Instance?.....	285
5.6 Monitoring and Alarm.....	285

5.6.1 Does Redis Support Command Audits?.....	286
5.6.2 What Should I Do If the Monitoring Data of a DCS Redis Instance Is Abnormal?.....	286
5.6.3 Why Is Available Memory of Unused DCS Instances Less Than Total Memory and Why Is Memory Usage of Unused DCS Instances Greater Than Zero?.....	286
5.6.4 Why Is Used Memory Greater Than Available Memory?.....	286
5.6.5 Why Is Flow Control Triggered? How Do I Handle It?.....	286
5.7 Data Backup, Export, and Migration.....	287
5.7.1 How Do I Export DCS Redis Instance Data?.....	287
5.7.2 Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?.....	288
5.7.3 Why Are Processes Frequently Killed During Data Migration?.....	288
5.7.4 Is All Data in a DCS Redis Instance Migrated During Online Migration?.....	288
5.7.5 Do DCS Redis Instances Support Data Persistence? What Is the Impact of Persistence?.....	288
5.7.6 When Will AOF Rewrites Be Triggered?.....	289
5.7.7 Online Migration with Rump.....	290
5.8 Big/Hot Key Analysis and Expired Key Scan.....	291
5.8.1 What Are Big Keys and Hot Keys?.....	291
5.8.2 What Is the Impact of Big Keys or Hot Keys?.....	292
5.8.3 How Do I Avoid Big Keys and Hot Keys?.....	293
5.8.4 How Do I Analyze the Hot Keys of a DCS Redis 3.0 Instance?.....	295
5.8.5 How Do I Detect Big Keys and Hot Keys in Advance?.....	295
5.8.6 How Does DCS Delete Expired Keys?.....	295
5.8.7 How Long Are Keys Stored? How Do I Set Key Expiration?.....	296
5.9 Master/Standby Switchover.....	296
5.9.1 When Does a Master/Standby Switchover Occur?.....	296
5.9.2 How Does Master/Standby Switchover Affect Services?.....	297
5.9.3 Does the Client Need to Switch the Connection Address After a Master/Standby Switchover?.....	297
5.9.4 How Does Redis Master/Standby Replication Work?.....	297

# 1 Service Overview

---

## 1.1 What Is DCS?

Distributed Cache Service (DCS) is an online, distributed, fast in-memory cache service compatible with Redis. It is reliable, scalable, usable out of the box, and easy to manage, meeting your requirements for high read/write performance and fast data access.

- Usability out of the box  
DCS provides single-node, master/standby, and cluster instances with specifications ranging from 128 MB to 1024 GB. DCS instances can be created with just a few clicks on the console, without requiring you to prepare servers. DCS Redis 4.0/5.0/6.0 instances are containerized and can be created within seconds.
- Security and reliability  
Instance data storage and access are securely protected through security management services, including Identity and Access Management (IAM), Virtual Private Cloud (VPC), and Cloud Eye.
- Auto scaling  
DCS instances can be scaled up or down online, helping you control costs based on service requirements.
- Easy management  
A web-based console is provided for you to perform various operations, such as restarting instances, modifying configuration parameters, and backing up and restoring data. RESTful application programming interfaces (APIs) are also provided for automatic instance management.
- Online migration  
You can create a data migration task on the console to import backup files or migrate data online.

### DCS for Redis

Redis is a storage system that supports multiple types of data structures, including key-value pairs. It can be used in such scenarios as data caching, event



publication/subscription, and high-speed queuing, as described in [Application Scenarios](#). Redis is written in ANSI C, supporting direct read/write of [strings](#), [hashes](#), [lists](#), [sets](#), [sorted sets](#), and [streams](#). Redis works with an in-memory dataset which can be persisted on disk.

DCS Redis instances can be customized based on your requirements.

**Table 1-1** DCS Redis instance configuration

<p><b>Instance type</b></p>	<p>DCS for Redis provides the following types of instances to suit different service scenarios:</p> <p>Single-node: Suitable for caching temporary data in low reliability scenarios. Single-node instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted.</p> <p>Master/standby: Each master/standby instance runs on two nodes (one master and one standby). The standby node replicates data synchronously from the master node. If the master node fails, the standby node automatically becomes the master node.</p> <p>Proxy Cluster: In addition to the native Redis cluster, a Proxy Cluster instance has proxies and load balancers. Load balancers implement load balancing. Different requests are distributed to different proxies to achieve high-concurrency. Each shard in the cluster has a master node and a standby node. If the master node is faulty, the standby node on the same shard is promoted to the master role to take over services.</p> <p>Redis Cluster: Each Redis Cluster instance consists of multiple <a href="#">shards</a> and each shard includes a master node and multiple replicas (or no replica at all). Shards are not visible to you. If the master node fails, a replica on the same shard takes over services. You can split read and write operations by writing to the master node and reading from the replicas. This improves the overall cache read/write performance.</p>
<p><b>Instance specification</b></p>	<p>DCS for Redis provides instances of different specifications, ranging from 128 MB to 1024 GB.</p>
<p><b>Redis version</b></p>	<p>DCS instances are compatible with open-source Redis 3.0/4.0/5.0/6.0.</p>
<p><b>Underlying architecture</b></p>	<p>Deployed on large-specs VMs. 50,000 QPS at a single node.</p>

For more information about open-source Redis, visit <https://redis.io/>.

## 1.2 Application Scenarios

### Redis Application Scenarios

Many large-scale e-commerce websites and video streaming and gaming applications require fast access to large amounts of data that has simple data structures and does not need frequent join queries. In such scenarios, you can use Redis to achieve fast yet inexpensive access to data. Redis enables you to retrieve data from in-memory data stores instead of relying entirely on slower disk-based databases. In addition, you no longer need to perform additional management tasks. These features make Redis an important supplement to traditional disk-based databases and a basic service essential for internet applications receiving high-concurrency access.

Typical application scenarios of DCS for Redis are as follows:

#### 1. E-commerce flash sales

E-commerce product catalogue, deals, and flash sales data can be cached to Redis.

For example, the high-concurrency data access in flash sales can be hardly handled by traditional relational databases. It requires the hardware to have higher configuration such as disk I/O. By contrast, Redis supports 50,000 QPS per node and allows you to implement locking using simple commands such as **SET**, **GET**, **DEL**, and **RPUSE** to handle flash sales.

#### 2. Live video commenting

In live streaming, online user, gift ranking, and bullet comment data can be stored as sorted sets in Redis.

For example, bullet comments can be returned using the **ZREVRANGEBYSCORE** command. The **ZPOPMAX** and **ZPOPMIN** commands in Redis 5.0 can further facilitate message processing.

#### 3. Game leaderboard

In online gaming, the highest ranking players are displayed and updated in real time. The leaderboard ranking can be stored as sorted sets, which are easy to use with up to 20 commands.

#### 4. Social networking comments

In web applications, queries of post comments often involve sorting by time in descending order. As comments pile up, sorting becomes less efficient.

By using lists in Redis, a preset number of comments can be returned from the cache, rather than from disk, easing the load off the database and accelerating application responses.

## 1.3 DCS Instance Types

### 1.3.1 Single-Node Redis

A single-node DCS Redis instance has only one node and does not support data persistence. They are suitable for cache services that do not require data reliability.

 NOTE

- DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. You are advised to use DCS Redis 4.0 and later instances.
- You cannot upgrade the Redis version of an instance. For example, a single-node DCS instance cannot be upgraded from Redis 4.0 to Redis 5.0. If you need Redis features of later versions, create a DCS Redis instance of a later version and then migrate data from the earlier instance to the new one.
- Single-node instances cannot ensure data persistence and do not support manual or scheduled data backup. Exercise caution before using them.

## Features

1. Low system overhead and high QPS

Single-node instances do not support data synchronization or data persistence, reducing system overhead and supporting higher concurrency. QPS of single-node DCS Redis instances reaches up to 50,000.

2. Process monitoring and automatic fault recovery

With an HA monitoring mechanism, if a single-node DCS instance becomes faulty, a new process is started within 30 seconds to resume service provisioning.

3. Out-of-the-box usability and no data persistence

Single-node DCS instances can be used out of the box because they do not involve data loading. If your service requires high QPS, you can warm up the data beforehand to avoid strong concurrency impact on the backend database.

4. Low-cost and suitable for development and testing

Single-node instances are 40% cheaper than master/standby DCS instances, suitable for setting up development or testing environments.

In summary, single-node DCS instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. They are suitable for scenarios which do not require data persistence, such as database front-end caching, to accelerate access and ease the concurrency load off the backend. If the desired data does not exist in the cache, requests will go to the database. When restarting the service or the DCS instance, you can pre-generate cache data from the disk database to relieve pressure on the backend during startup.

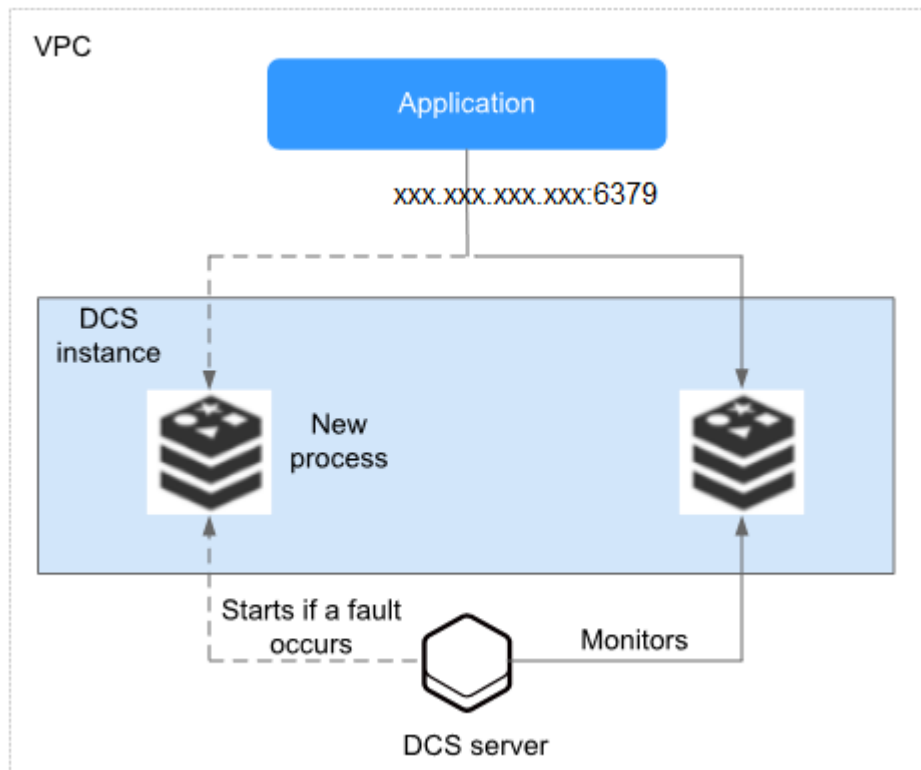
## Architecture

**Figure 1-1** shows the architecture of single-node DCS Redis instances.

 NOTE

To access a DCS Redis 3.0 instance, you must use port 6379. To access a DCS Redis 4.0 or later instance, you can customize the port. If no port is specified, the default port 6379 will be used. In the following architecture, port 6379 is used. If you have customized a port, replace **6379** with the actual port.

**Figure 1-1** Single-node DCS Redis instance architecture



Architecture description:

- **VPC**  
All server nodes of the instance run in the same VPC.  
  
**NOTE**  
For intra-VPC access, the client and the instance must be in the same VPC.
- **Application**  
The client of the instance, which is the application running on an Elastic Cloud Server (ECS).  
DCS Redis instances are compatible with the Redis protocol, and can be accessed through open-source clients. For details about accessing DCS instances, see *Distributed Cache Service Developer Guide* > "Accessing an Instance".
- **DCS instance**  
A single-node DCS instance, which has only one node and one Redis process.  
DCS monitors the availability of the instance in real time. If the Redis process becomes faulty, DCS starts a new process to resume service provisioning.

### 1.3.2 Master/Standby Redis

This section describes master/standby DCS Redis instances.

 **NOTE**

DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. You are advised to use DCS Redis 4.0 and later instances.

You cannot upgrade the Redis version for an instance. For example, a master/standby DCS Redis 4.0 instance cannot be upgraded to a master/standby DCS Redis 5.0 instance. If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the old instance to the new one.

## Features

Master/Standby DCS instances have higher availability and reliability than single-node DCS instances.

Master/Standby DCS instances have the following features:

1. **Data persistence and high reliability**

By default, data persistence is enabled by both the master and the standby node of a master/standby instance.

The standby node of a DCS Redis 3.0 instance is invisible to you. Only the master node provides data read/write operations.

The standby node of a Redis 4.0 or later instance is visible to you. You can read data from the standby node by connecting to it using the instance read-only address.

2. **Data synchronization**

Data in the master and standby nodes is kept consistent through incremental synchronization.

 **NOTE**

After recovering from a network exception or node fault, master/standby instances perform a full synchronization to ensure data consistency.

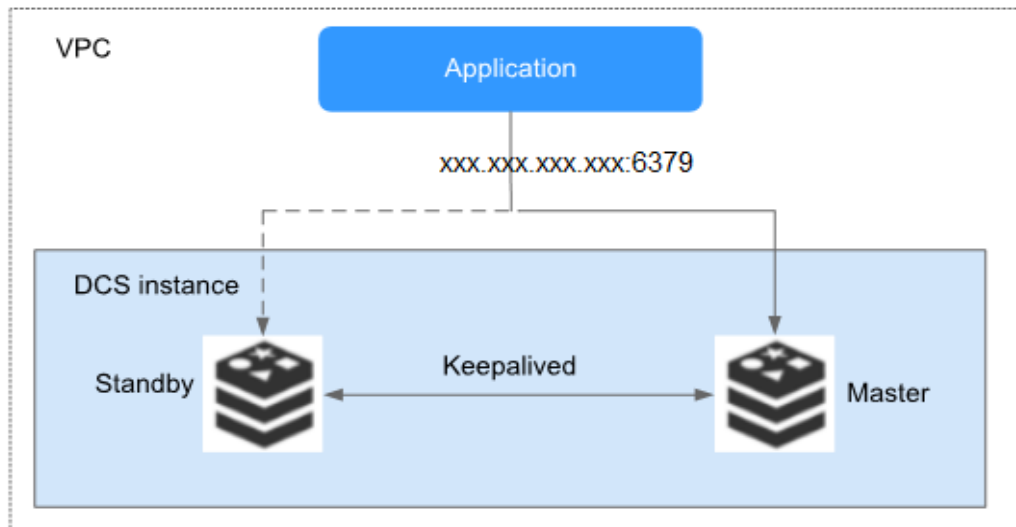
3. **Automatic master/standby switchover**

If the master node becomes faulty, the instance is disconnected and unavailable for several seconds. The standby node takes over within 30 seconds without manual operations to resume stable services.

## Architecture of DCS Redis 3.0 Instances

[Figure 1-2](#) shows the architecture of master/standby DCS Redis instances.

**Figure 1-2** Master/Standby DCS instance architecture



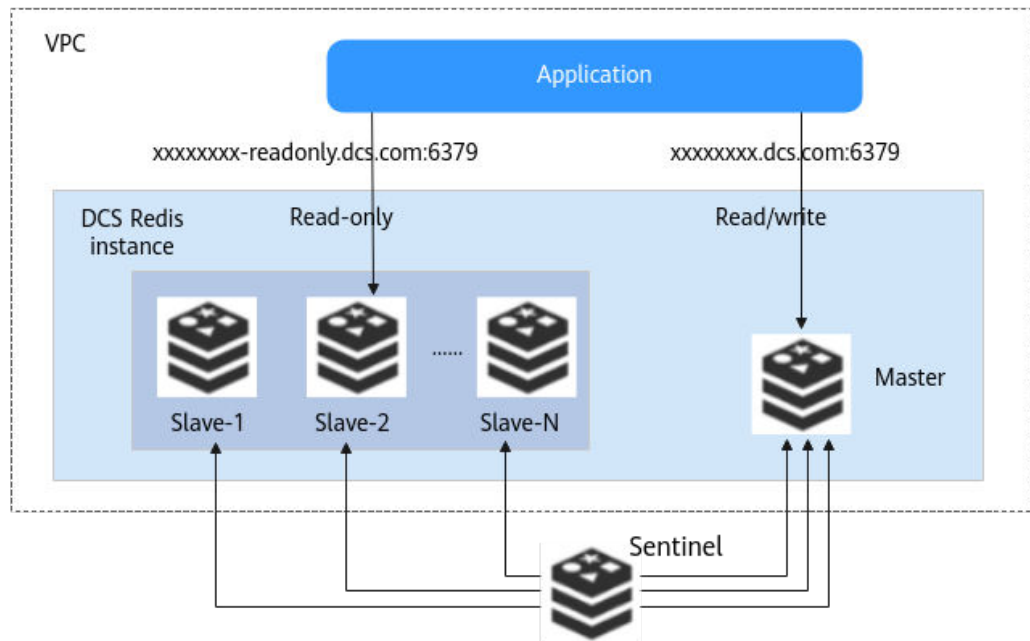
Architecture description:

- **VPC**  
All server nodes of the instance run in the same VPC.  
**NOTE**  
For intra-VPC access, the client and the instance must be in the same VPC.
- **Application**  
The Redis client of the instance, which is the application running on the ECS. DCS Redis instances are compatible with the Redis protocol, and can be accessed through open-source clients. For details about accessing DCS instances, see *Distributed Cache Service Developer Guide* > "Accessing an Instance".
- **DCS instance**  
Indicates a master/standby DCS instance which has a master node and a standby node. By default, data persistence is enabled and data is synchronized between the two nodes.  
DCS monitors the availability of the instance in real time. If the master node becomes faulty, the standby node becomes the master node and resumes service provisioning.  
DCS Redis 3.0 instances are accessed through port 6379 by default. Port customization is not supported.

## Architecture of Master/Standby DCS Redis 4.0/5.0/6.0 Instances

The following figure shows the architecture of a master/standby DCS Redis 4.0/5.0/6.0 instance.

**Figure 1-3** Architecture of a master/standby DCS Redis 4.0/5.0/6.0 instance



Architecture description:

1. Each master/standby DCS Redis 4.0 or later instance has two connection addresses. When connecting to such an instance, you can use the read/write domain name address to connect to the master node or use the read-only domain name address to connect to the standby node.  
The connection addresses can be obtained on the instance details page on the DCS console.
2. Master/Standby DCS Redis 4.0 and later instances support Sentinels. Sentinels monitor the running status of the master and standby nodes. If the master node becomes faulty, a failover will be performed.  
Sentinels are invisible to you and is used only in the service.
3. A standby node has the same specifications as a master node. A master/standby instance consists of a pair of master and standby nodes by default.
4. To access a DCS Redis 4.0 or later instance, you can customize the port. If no port is specified, the default port 6379 will be used. In the architecture diagram, port 6379 is used. If you have customized a port, replace **6379** with the actual port.

**NOTE**

To implement read/write splitting using a master/standby instance, ensure that your client can distinguish between read and write requests. The client directs write requests to the read/write domain name and read requests to the read-only domain name.

**Requests to the read-only domain name address may fail if the standby node of a master/standby DCS Redis 4.0 or later instance is faulty. For higher reliability and lower latency, do not use the read-only address.**

### 1.3.3 Proxy Cluster Redis

DCS for Redis provides Proxy Cluster instances, which use Linux Virtual Server (LVS) and proxies to achieve high availability. Proxy Cluster instances have the following features:

- The client is decoupled from the cloud service.
- They support millions of concurrent requests, equivalent to Redis Cluster instances.
- A wide range of memory specifications adapt to different scenarios.

 **NOTE**

- A Proxy Cluster instance can be connected in the same way that a single-node or master/standby instance is connected, without any special settings on the client. You can use the IP address or domain name of the instance, and do not need to know or use the proxy or shard addresses.
- You cannot upgrade the Redis version for an instance. For example, a Proxy Cluster DCS Redis 4.0 instance cannot be upgraded to a Proxy Cluster DCS Redis 5.0 instance. If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the old instance to the new one.
- DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. DCS Redis 4.0 or 5.0 instances are recommended.
- Redis 4.0 and 5.0 depend on ELB.

#### Proxy Cluster DCS Redis 3.0 Instances

Proxy Cluster DCS Redis 3.0 instances are compatible with [codis](#). The specifications range from 64 GB to 1024 GB, meeting requirements for **millions of concurrent connections** and **massive data cache**. Distributed data storage and access is implemented by DCS, without requiring development or maintenance.

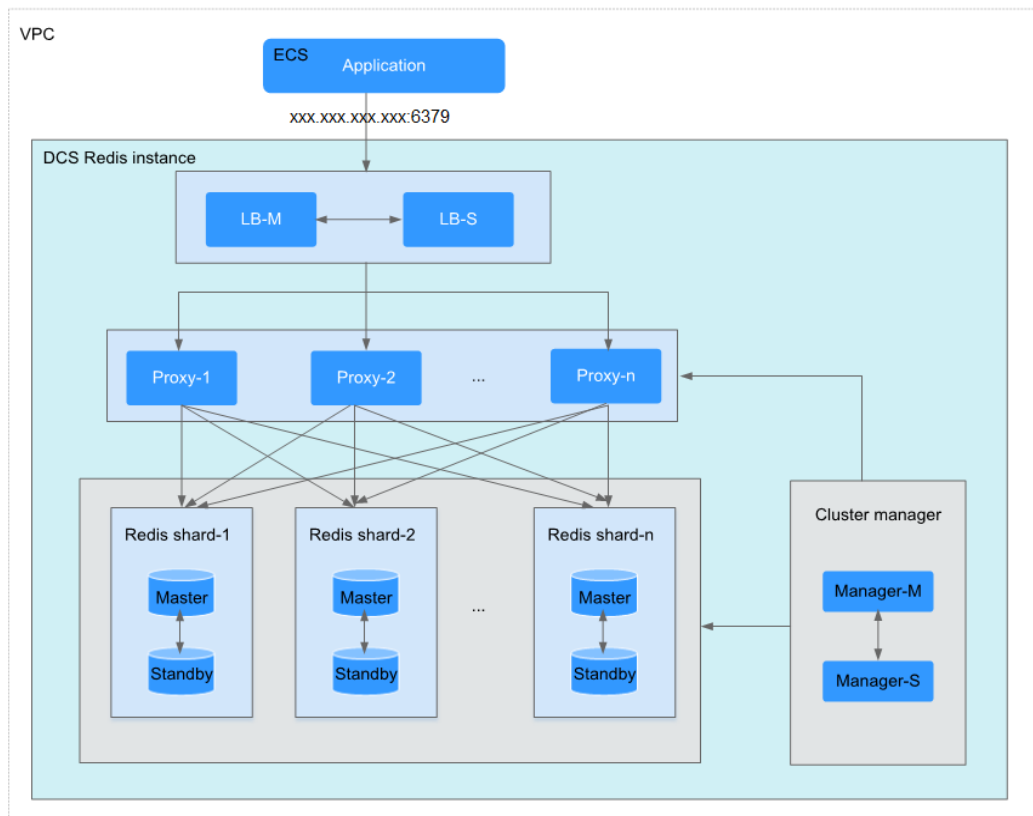
Each Proxy Cluster instance consists of load balancers, proxies, cluster managers, and [shards](#).

**Table 1-2** Total memory, proxies, and shards of Proxy Cluster DCS Redis 3.0 instances

Total Memory	Proxies	Shards
64 GB	3	8
128 GB	6	16
256 GB	8	32



**Figure 1-4** Proxy Cluster DCS Redis instance architecture



Architecture description:

- **VPC**

All server nodes of the cluster instance run in the same VPC.

**NOTE**

For intra-VPC access, the client and the instance must be in the same VPC.

- **Application**

The client used to access the instance.

DCS Redis instances can be accessed through open-source clients. For details about accessing DCS instances, see *Distributed Cache Service Developer Guide* > "Accessing an Instance".

- **LB-M/LB-S**

The load balancers, which are deployed in master/standby HA mode. The connection addresses (**IP address:Port**) of the cluster DCS Redis instance are the addresses of the load balancers.

- **Proxy**

The proxy server used to achieve high availability and process high-concurrency client requests.

You can connect to a Proxy Cluster instance at the IP addresses of its proxies.

- **Redis shard**

A shard of the cluster.

Each shard consists of a pair of master/standby nodes. If the master node becomes faulty, the standby node automatically takes over cluster services.

If both the master and standby nodes of a shard are faulty, the cluster can still provide services but the data on the faulty shard is inaccessible.

- **Cluster manager**

The cluster configuration managers, which store configurations and partitioning policies of the cluster. You cannot modify the information about the configuration managers.

## Proxy Cluster DCS Redis 4.0 and 5.0 Instances

Proxy Cluster DCS Redis 4.0 and 5.0 instances are built based on open-source Redis 4.0 and 5.0 and compatible with [open source codis](#). They provide multiple large-capacity specifications ranging from 4 GB to 1024 GB and support the x86 and Arm CPU architectures.

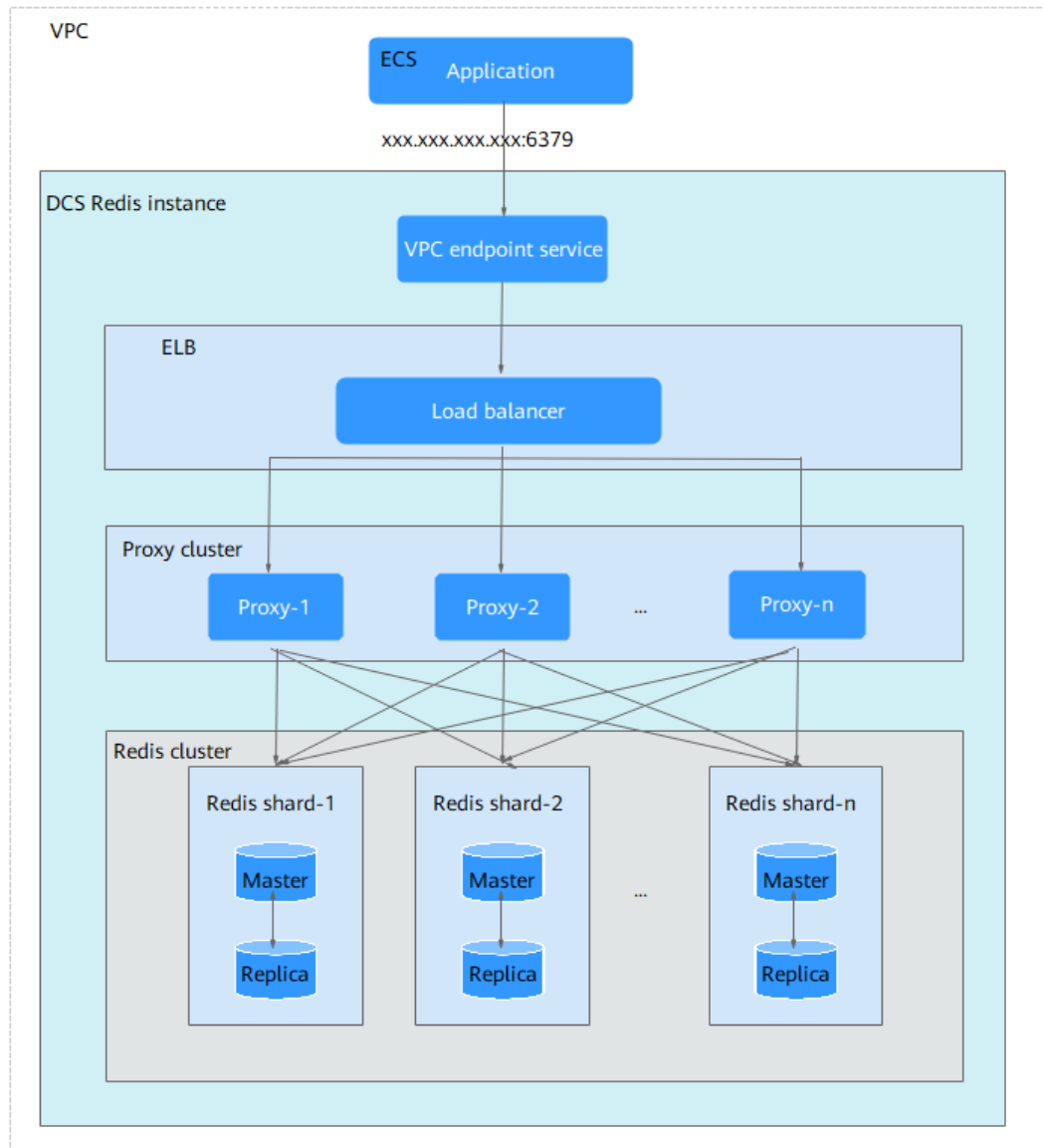
**Table 1-3** lists the number of shards corresponding to different specifications. You can customize the shard size when creating an instance. Currently, the number of shards and replicas cannot be customized. By default, each shard has two replicas.

**Memory per shard = Instance specification/Number of shards.** For example, if a 48 GB instance has 6 shards, the size of each shard is  $48 \text{ GB}/6 = 8 \text{ GB}$ .

**Table 1-3** Specifications of Proxy Cluster DCS Redis 4.0 and 5.0 instances

Total Memory	Proxies	Shards	Memory per Shard (GB)
4 GB	3	3	1.33
8 GB	3	3	2.67
16 GB	3	3	5.33
24 GB	3	3	8
32 GB	3	3	10.67
48 GB	6	6	8
64 GB	8	8	8
96 GB	12	12	8
128 GB	16	16	8
192 GB	24	24	8
256 GB	32	32	8
384 GB	48	48	8
512 GB	64	64	8
768 GB	96	96	8
1024 GB	128	128	8

**Figure 1-5** Architecture of a Proxy Cluster DCS Redis 4.0 or 5.0 instance



Architecture description:

- **VPC**  
All server nodes of the cluster instance run in the same VPC.
- **Application**  
The client used to access the instance.  
DCS Redis instances can be accessed through open-source clients. For details about accessing DCS instances in different languages, see *Distributed Cache Service Developer Guide* > "Accessing an Instance".
- **VPC endpoint service**  
You can configure your DCS Redis instance as a VPC endpoint service and access the instance at the VPC endpoint service address.  
The IP address of the Proxy Cluster DCS Redis instance is the address of the VPC endpoint service.

- **ELB**  
The load balancers, which are deployed in cluster HA mode.
- **Proxy**  
The proxy server used to achieve high availability and process high-concurrency client requests.  
You cannot connect to a Proxy Cluster instance at the IP addresses of its proxies.
- **Redis cluster**  
A shard of the cluster.  
Each shard consists of a pair of master/replica nodes. If the master node becomes faulty, the replica node automatically takes over cluster services.  
If both the master and standby nodes of a shard are faulty, the cluster can still provide services but the data on the faulty shard is inaccessible.

### 1.3.4 Redis Cluster

Redis Cluster DCS instances use the native distributed implementation of Redis. Redis Cluster instances have the following features:

- They are compatible with native Redis clusters.
- They inherit the smart client design from Redis.
- They deliver many times higher performance than master/standby instances.

#### Redis Cluster

The Redis Cluster instance type provided by DCS is compatible with the **native Redis Cluster**, which uses smart clients and a distributed architecture to perform sharding.

**Table 1-4** lists the shard specification for different instance specifications.

**Size of a shard = Instance specification/Number of shards.** For example, if a 48 GB instance has 6 shards, the size of each shard is  $48 \text{ GB}/6 = 8 \text{ GB}$ .

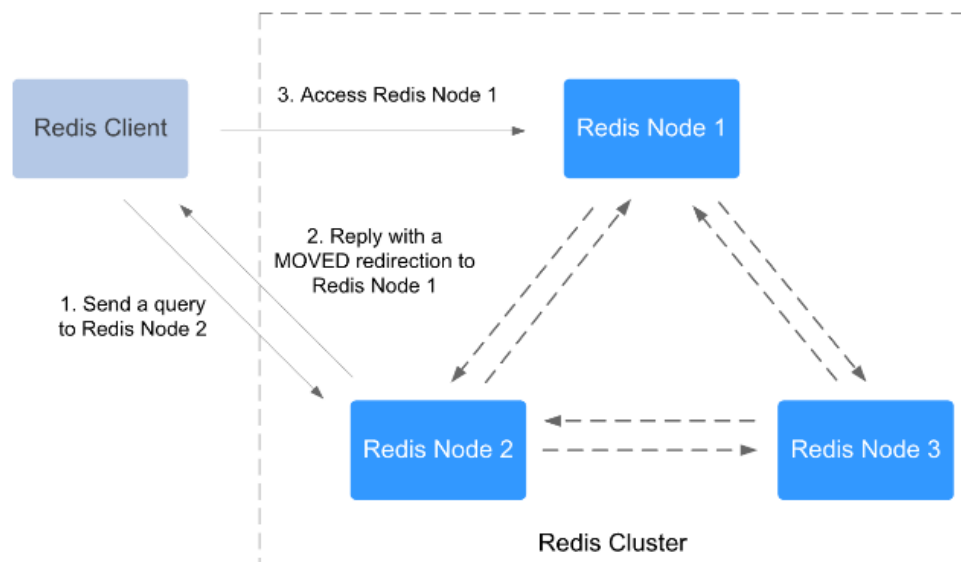
**Table 1-4** Specifications of Redis Cluster DCS instances

Total Memory	Shards
4 GB/8 GB/16 GB/24 GB/32 GB	3
48 GB	6
64 GB	8
96 GB	12
128 GB	16
192 GB	24
256 GB	32
384 GB	48

Total Memory	Shards
512 GB	64
768 GB	96
1024 GB	128

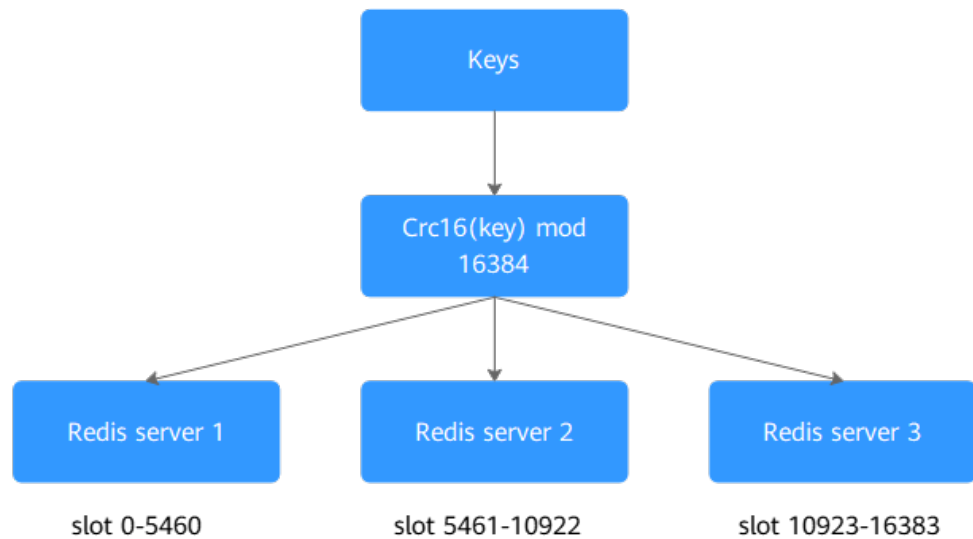
- Distributed architecture**  
 Any node in a Redis Cluster can receive requests. Received requests are then redirected to the right node for processing. Each node consists of a subset of one master and one (by default) or multiple replicas. The master or replica roles are determined through an election algorithm.

**Figure 1-6** Distributed architecture of Redis Cluster



- Presharding**  
 There are 16,384 hash slots in each Redis Cluster. The mapping between hash slots and Redis nodes is stored in Redis Servers. To compute what is the hash slot of a given key, simply take the CRC16 of the key modulo 16384. Example command output

Figure 1-7 Redis Cluster presharding



## 1.4 DCS Instance Specifications

### 1.4.1 Redis 3.0 Instance Specifications (Obsolete)

This section describes DCS Redis 3.0 instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- **Used memory:** You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- **Maximum connections:** The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.
- **QPS** represents queries per second, which is the number of commands processed per second.

#### NOTE

- Single-node, master/standby, and Proxy Cluster types are available.
- DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. DCS Redis 4.0 or 5.0 instances are recommended.
- Both x86 and Arm architectures are supported.

### Single-Node Instances

For each single-node DCS Redis instance, the available memory is less than the total memory because some memory is reserved for system overheads, as shown in the following table.

**Table 1-5** Specifications of single-node DCS Redis 3.0 instances

CPU	Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
Arm	2	1.2	5000/5000	42/512	50,000	dcs.arm.single_node
	4	2.4	5000/5000	64/1536	50,000	
	8	4.8	5000/5000	64/1536	50,000	
	16	9.6	5000/5000	85/3072	50,000	
	32	19.2	5000/5000	85/3072	50,000	
	64	38.4	5000/6000	128/5120	50,000	
x86	2	1.5	5000/50,000	42/512	50,000	dcs.single_node
	4	3.2	5000/50,000	64/1536	50,000	
	8	6.8	5000/50,000	64/1536	50,000	
	16	13.6	5000/50,000	85/3072	50,000	
	32	27.2	5000/50,000	85/3072	50,000	
	64	58.2	5000/60,000	128/5120	50,000	

## Master/Standby Instances

For each master/standby DCS Redis instance, the available memory is less than that of a single-node DCS Redis instance because some memory is reserved for data persistence, as shown in the following table. The available memory of a master/standby instance can be adjusted to support background tasks such as data persistence and master/standby synchronization.

**Table 1-6** Specifications of master/standby DCS Redis 3.0 instances

CPU	Total Memory (GB)	Available Memory (GB)	Maximum Connections Allowed (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
Arm	2	1.2	5000/5000	42/512	50,000	dcs.arm.master_standby
	4	2.4	5000/5000	64/1536	50,000	
	8	4.8	5000/5000	64/1536	50,000	

CPU	Total Memory (GB)	Available Memory (GB)	Maximum Connections Allowed (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
	16	9.6	5000/5000	85/3072	50,000	
	32	19.2	5000/5000	85/3072	50,000	
	64	38.4	5000/6000	128/5120	50,000	
x86	2	1.5	5000/50,000	42/512	50,000	dcs.master_standby
	4	3.2	5000/50,000	64/1536	50,000	
	8	6.4	5000/50,000	64/1536	50,000	
	16	12.8	5000/50,000	85/3072	50,000	
	32	25.6	5000/50,000	85/3072	50,000	
	64	51.2	5000/60,000	128/5120	50,000	

## Proxy Cluster Instances

In addition to larger memory, cluster instances feature more connections allowed, higher bandwidth allowed, and more QPS than single-node and master/standby instances.

**Table 1-7** Specifications of Proxy Cluster DCS Redis 3.0 instances

CPU	Specification (GB)	Available Memory (GB)	Maximum Connections Allowed (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
Arm	64	64	30,000/30,000	600/5120	100,000	dcs.arm.cluster
	128	128	60,000/60,000	600/5120	100,000	



CPU	Specification (GB)	Available Memory (GB)	Maximum Connections Allowed (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
	256	256	60,000/60,000	600/5120	100,000	
x86	64	64	90,000/90,000	600/5120	100,000	dcs.cluster
	128	128	180,000/180,000	600/5120	100,000	
	256	256	240,000/240,000	600/5120	100,000	

## 1.4.2 Redis 4.0 and 5.0 Instance Specifications

This section describes DCS Redis 4.0 and 5.0 instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- **Used memory:** You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- **Maximum connections:** The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric. After an instance is created, you can change the maximum number of connections of the instance by modifying the **maxclients** parameter on the **Instance Configuration > Parameters** page on the console. This parameter cannot be modified for Proxy Cluster instances.
- **QPS** represents queries per second, which is the number of commands processed per second.
- **Bandwidth:** You can view the **Flow Control Times** metric to check whether the bandwidth has exceeded the limit.

### NOTE

- Single-node, master/standby, Proxy Cluster, and Redis Cluster instance types are available.
- Supported CPU architecture: x86 and Arm.

## Single-Node Instances

**Table 1-8** Specifications of single-node DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	50,000	x86: redis.single.xu1.tiny.128 Arm: redis.single.au1.tiny.128
0.25	0.25	10,000/10,000	80/80	50,000	x86: redis.single.xu1.tiny.256 Arm: redis.single.au1.tiny.256
0.5	0.5	10,000/10,000	80/80	50,000	x86: redis.single.xu1.tiny.512 Arm: redis.single.au1.tiny.512
1	1	10,000/10,000	80/80	50,000	x86: redis.single.xu1.large.1 Arm: redis.single.au1.large.1
2	2	10,000/10,000	128/128	50,000	x86: redis.single.xu1.large.2 Arm: redis.single.au1.large.2

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	10,000/10,000	192/192	50,000	x86: redis.single.xu1.large.4 Arm: redis.single.au1.large.4
8	8	10,000/10,000	192/192	50,000	x86: redis.single.xu1.large.8 Arm: redis.single.au1.large.8
16	16	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.16 Arm: redis.single.au1.large.16
24	24	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.24 Arm: redis.single.au1.large.24
32	32	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.32 Arm: redis.single.au1.large.32
48	48	10,000/10,000	256/256	50,000	x86: redis.single.xu1.large.48 Arm: redis.single.au1.large.48

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
64	64	10,000/10,000	384/384	50,000	x86: redis.single.xu1.large.64 Arm: redis.single.au1.large.64

## Master/Standby Instances

By default, a master/standby instance has two replicas (including the master). There is one master node.

Number of IP addresses occupied by a master/standby instance = Number of master nodes × Number of replicas. For example:

2 replicas: Number of occupied IP addresses = 1 × 2 = 2

3 replicas: Number of occupied IP addresses = 1 × 3 = 3

The following table lists the specification codes (**spec\_code**) when there are two default replicas. Change the replica quantity in the specification codes based on the actual number of replicas. For example, if an 8 GB master/standby x86-based instance has two replicas, its specification code is redis.ha.xu1.large. **r2.8**. If it has three replicas, its specification code is redis.ha.xu1.large. **r3.8**.

**Table 1-9** Specifications of master/standby DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	50,000	x86: redis.ha.xu1.tiny.r2.128 Arm: redis.ha.au1.tiny.r2.128

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.25	0.25	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.tiny.r2.256 Arm: redis.ha.au1.tiny.r2.256
0.5	0.5	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.tiny.r2.512 Arm: redis.ha.au1.tiny.r2.512
1	1	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.large.r2.1 Arm: redis.ha.au1.large.r2.1
2	2	10,000/10,000	128/128	50,000	x86: redis.ha.xu1.large.r2.2 Arm: redis.ha.au1.large.r2.2
4	4	10,000/10,000	192/192	50,000	x86: redis.ha.xu1.large.r2.4 Arm: redis.ha.au1.large.r2.4
8	8	10,000/10,000	192/192	50,000	x86: redis.ha.xu1.large.r2.8 Arm: redis.ha.au1.large.r2.8

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
16	16	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.16 Arm: redis.ha.au1.large.r2.16
24	24	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.24 Arm: redis.ha.au1.large.r2.24
32	32	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.32 Arm: redis.ha.au1.large.r2.32
48	48	10,000/10,000	256/256	50,000	x86: redis.ha.xu1.large.r2.48 Arm: redis.ha.au1.large.r2.48
64	64	10,000/10,000	384/384	50,000	x86: redis.ha.xu1.large.r2.64 Arm: redis.ha.au1.large.r2.64

## Proxy Cluster Instances

The number of shards and replicas of a Proxy Cluster instance cannot be customized. By default, each shard has two replicas. For details about the default number of shards, see [Table 1-3](#).

**Table 1-10** Specifications of Proxy Cluster DCS Redis 4.0 and 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	10,000/10,000	1000/1000	100,000	x86: redis.proxy.xu1.large.4 Arm: redis.proxy.au1.large.4
8	8	10,000/10,000	2000/2000	100,000	x86: redis.proxy.xu1.large.8 Arm: redis.proxy.au1.large.8
16	16	10,000/10,000	3072/3072	100,000	x86: redis.proxy.xu1.large.16 Arm: redis.proxy.au1.large.16
24	24	10,000/10,000	3072/3072	100,000	x86: redis.proxy.xu1.large.24 Arm: redis.proxy.au1.large.24
32	32	10,000/10,000	3072/3072	100,000	x86: redis.proxy.xu1.large.32 Arm: redis.proxy.au1.large.32
48	48	10,000/10,000	4608/4608	200,000	x86: redis.proxy.xu1.large.48 Arm: redis.proxy.au1.large.48

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
64	64	10,000/10,000	6144/6144	250,000	x86: redis.proxy.xu1.large.64 Arm: redis.proxy.au1.large.64
96	96	10,000/10,000	9216/9216	400,000	x86: redis.proxy.xu1.large.96 Arm: redis.proxy.au1.large.96
128	128	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.128 Arm: redis.proxy.au1.large.128
192	192	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.192 Arm: redis.proxy.au1.large.192
256	256	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.256 Arm: redis.proxy.au1.large.256
384	384	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.384 Arm: redis.proxy.au1.large.384



Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
512	512	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.512 Arm: redis.proxy.au1.large.512
768	768	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.768 Arm: redis.proxy.au1.large.768
1024	1024	10,000/10,000	10,000/10,000	500,000	x86: redis.proxy.xu1.large.1024 Arm: redis.proxy.au1.large.1024

## Redis Cluster Instances

In addition to larger memory, Redis Cluster instances feature more connections allowed, higher bandwidth allowed, and more QPS than single-node and master/standby instances.

- Specification name: The following table only lists the specification names of 2-replica x86- and Arm-based instances. The specification names reflect the number of replicas, for example, redis.cluster.xu1.large.r2.8 (x86 | 2 replicas | 8 GB) and redis.cluster.xu1.large.r3.8 (x86 | 3 replicas | 8 GB).
- IP addresses: Number of occupied IP addresses = Number of shards × Number of replicas. For example:  
4 GB | Redis Cluster | 3 replicas | 3 shards: Number of occupied IP addresses = 3 × 3 = 9
- Available memory per node = Instance available memory/Master node quantity. For example:  
A 24 GB instance has 24 GB available memory and 3 master nodes. The available memory per node is 24/3 = 8 GB.
- Maximum connections per node = Instance maximum connections/Master node quantity. For example:

A 4 GB instance has 3 master nodes and the maximum connections limit is 150,000. The maximum connections limit per node =  $150,000/3 = 50,000$ .

**Table 1-11** Specifications of Redis Cluster DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	3	10,000/10,000	2304/2304	100,000	x86: redis.cluster.xu1.large.r2.4 Arm: redis.cluster.au1.large.r2.4
8	8	3	10,000/10,000	2304/2304	100,000	x86: redis.cluster.xu1.large.r2.8 Arm: redis.cluster.au1.large.r2.8
16	16	3	10,000/10,000	2304/2304	100,000	x86: redis.cluster.xu1.large.r2.16 Arm: redis.cluster.au1.large.r2.16
24	24	3	10,000/10,000	2304/2304	100,000	x86: redis.cluster.xu1.large.r2.24 Arm: redis.cluster.au1.large.r2.24

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
32	32	3	10,000/10,000	2304/2304	100,000	x86: redis.cluster.xu1.large.r2.32 Arm: redis.cluster.au1.large.r2.32
48	48	6	10,000/10,000	4608/4608	200,000	x86: redis.cluster.xu1.large.r2.48 Arm: redis.cluster.au1.large.r2.48
64	64	8	10,000/10,000	6144/6144	250,000	x86: redis.cluster.xu1.large.r2.64 Arm: redis.cluster.au1.large.r2.64
96	96	12	10,000/10,000	9216/9216	400,000	x86: redis.cluster.xu1.large.r2.96 Arm: redis.cluster.au1.large.r2.96

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
128	128	16	10,000/10,000	12,288/12,288	500,000	x86: redis.cluster.xu1.large.r2.128 Arm: redis.cluster.au1.large.r2.128
192	192	24	10,000/10,000	18,432/18,432	500,000	x86: redis.cluster.xu1.large.r2.192 Arm: redis.cluster.au1.large.r2.192
256	256	32	10,000/10,000	24,576/24,576	500,000	x86: redis.cluster.xu1.large.r2.256 Arm: redis.cluster.au1.large.r2.256
384	384	48	10,000/10,000	36,864/36,864	500,000	x86: redis.cluster.xu1.large.r2.384 Arm: redis.cluster.au1.large.r2.384

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
512	512	64	10,000/10,000	49,152/49,152	500,000	x86: redis.cluster.xu1.large.r2.512 Arm: redis.cluster.au1.large.r2.512
768	768	96	10,000/10,000	73,728/73,728	500,000	x86: redis.cluster.xu1.large.r2.768 Arm: redis.cluster.au1.large.r2.768
1024	1024	128	10,000/10,000	98,304/98,304	500,000	x86: redis.cluster.xu1.large.r2.1024 Arm: redis.cluster.au1.large.r2.1024

### 1.4.3 Redis 6.0 Instance Specifications

This section describes DCS Redis 6.0 instance specifications, including the total memory, available memory, maximum number of connections, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- **Used memory:** You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- **Maximum connections:** The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.

- QPS represents queries per second, which is the number of commands processed per second.
- Bandwidth: You can view the **Flow Control Times** metric to check whether the bandwidth has exceeded the limit.

Currently, DCS for Redis 6.0 supports single-node and master/standby single-node, master/standby, and Redis Cluster instances.

## Single-Node

**Table 1-12** Specifications of single-node DCS Redis 6.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	50,000	x86: redis.single.xu1.tiny.128 Arm: redis.single.au1.tiny.128
0.25	0.25	10,000/10,000	80/80	50,000	x86: redis.single.xu1.tiny.256 Arm: redis.single.au1.tiny.256
0.5	0.5	10,000/10,000	80/80	50,000	x86: redis.single.xu1.tiny.512 Arm: redis.single.au1.tiny.512
1	1	10,000/50,000	80/80	50,000	x86: redis.single.xu1.large.1 Arm: redis.single.au1.large.1
2	2	10,000/50,000	128/128	50,000	x86: redis.single.xu1.large.2 Arm: redis.single.au1.large.2

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	10,000/50,000	192/192	50,000	x86: redis.single.xu1.large.4 Arm: redis.single.au1.large.4
8	8	10,000/50,000	192/192	50,000	x86: redis.single.xu1.large.8 Arm: redis.single.au1.large.8
16	16	10,000/50,000	256/256	50,000	x86: redis.single.xu1.large.16 Arm: redis.single.au1.large.16
24	24	10,000/50,000	256/256	50,000	x86: redis.single.xu1.large.24 Arm: redis.single.au1.large.24
32	32	10,000/50,000	256/256	50,000	x86: redis.single.xu1.large.32 Arm: redis.single.au1.large.32
48	48	10,000/50,000	256/256	50,000	x86: redis.single.xu1.large.48 Arm: redis.single.au1.large.48

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
64	64	10,000/50,000	384/384	50,000	x86: redis.single.xu1.large.64  Arm: redis.single.au1.large.64

## Master/Standby

**Table 1-13** Specifications of master/standby DCS Redis 6.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	50,000	x86: redis.ha.xu1.tiny.r2.128  Arm: redis.ha.au1.tiny.r2.128
0.25	0.25	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.tiny.r2.256  Arm: redis.ha.au1.tiny.r2.256
0.5	0.5	10,000/10,000	80/80	50,000	x86: redis.ha.xu1.tiny.r2.512  Arm: redis.ha.au1.tiny.r2.512



Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
1	1	10,000/50,000	80/80	50,000	x86: redis.ha.xu1.large.r2.1 Arm: redis.ha.au1.large.r2.1
2	2	10,000/50,000	128/128	50,000	x86: redis.ha.xu1.large.r2.2 Arm: redis.ha.au1.large.r2.2
4	4	10,000/50,000	192/192	50,000	x86: redis.ha.xu1.large.r2.4 Arm: redis.ha.au1.large.r2.4
8	8	10,000/50,000	192/192	50,000	x86: redis.ha.xu1.large.r2.8 Arm: redis.ha.au1.large.r2.8
16	16	10,000/50,000	256/256	50,000	x86: redis.ha.xu1.large.r2.16 Arm: redis.ha.au1.large.r2.16
24	24	10,000/50,000	256/256	50,000	x86: redis.ha.xu1.large.r2.24 Arm: redis.ha.au1.large.r2.24

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
32	32	10,000/50,000	256/256	50,000	x86: redis.ha.xu1.large.r2.32 Arm: redis.ha.au1.large.r2.32
48	48	10,000/50,000	256/256	50,000	x86: redis.ha.xu1.large.r2.48 Arm: redis.ha.au1.large.r2.48
64	64	10,000/50,000	384/384	50,000	x86: redis.ha.xu1.large.r2.64 Arm: redis.ha.au1.large.r2.64

## Redis Cluster

- Specification code: [Table 1-14](#) only lists the specification codes of instances with 2 replicas (default). The specification names reflect the number of replicas, for example, redis.cluster.xu1.large.r2.4 (2 replicas | 4 GB) and redis.cluster.xu1.large.r1.4 (1 replica | 4 GB).
- **Number of used IP addresses:** Number of shards × Number of replicas. For example:  
8 GB | Redis Cluster | 2 replicas | 3 shards: Number of occupied IP addresses =  $3 \times 2 = 6$
- **Available memory per node:** Instance available memory/Master node quantity. For example:  
A 64 GB instance has 64 GB available memory and 8 master nodes. The available memory per node is  $64/8 = 8$  GB.
- **Max. connections per node:** Number of instance max. connections/Master node quantity. For example:  
A 24 GB instance has 3 master nodes and the maximum connections limit is 150,000. The maximum connections limit per node =  $150,000/3 = 50,000$ .

 NOTE

The maximum bandwidth and assured bandwidth as shown in the table below is for the entire instance, and not for a single shard. The relationship between the instance bandwidth and the bandwidth of a single shard is as follows:

- Instance bandwidth = Bandwidth of a single shard × Number of shards
- For a cluster instance, if the memory per shard is 1 GB, the bandwidth per shard is 384 Mbit/s. If the memory per shard is greater than 1 GB, the bandwidth per shard is 768 Mbit/s.

For example, if a 24 GB Redis Cluster instance has three shards and the memory per shard is 24/3 = 8 GB (greater than 1 GB), the bandwidth per shard is 768 Mbit/s.

**Table 1-14** Specifications of Redis Cluster DCS Redis 6.0 instances

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	3	30,000/150,000	2,304/2,304	100,000	x86: redis.cluster.xu1.l arge.r2.4 Arm: redis.cluster.au1.l arge.r2.4
8	8	3	30,000/150,000	2,304/2,304	100,000	x86: redis.cluster.xu1.l arge.r2.8 Arm: redis.cluster.au1.l arge.r2.8
16	16	3	30,000/150,000	2,304/2,304	100,000	x86: redis.cluster.xu1.l arge.r2.16 Arm: redis.cluster.au1.l arge.r2.16
24	24	3	30,000/150,000	2,304/2,304	100,000	x86: redis.cluster.xu1.l arge.r2.24 Arm: redis.cluster.au1.l arge.r2.24

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
32	32	3	30,000/150,000	2,304/2,304	100,000	x86: redis.cluster.xu1.l arge.r2.32 Arm: redis.cluster.au1.l arge.r2.32
48	48	6	60,000/300,000	4,608/4,608	200,000	x86: redis.cluster.xu1.l arge.r2.48 Arm: redis.cluster.au1.l arge.r2.48
64	64	8	80,000/400,000	6,144/6,144	250,000	x86: redis.cluster.xu1.l arge.r2.64 Arm: redis.cluster.au1.l arge.r2.64
96	96	12	120,000/600,000	9,216/9,216	400,000	x86: redis.cluster.xu1.l arge.r2.96 Arm: redis.cluster.au1.l arge.r2.96
128	128	16	160,000/800,000	12,288/12,288	500,000	x86: redis.cluster.xu1.l arge.r2.128 Arm: redis.cluster.au1.l arge.r2.128
192	192	24	240,000/1,200,000	18,432/18,432	500,000	x86: redis.cluster.xu1.l arge.r2.192 Arm: redis.cluster.au1.l arge.r2.192

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
256	256	32	320,000/1,600,000	24,576/24,576	500,000	x86: redis.cluster.xu1.l arge.r2.256 Arm: redis.cluster.au1.l arge.r2.256
384	384	48	480,000/2,400,000	36,864/36,864	500,000	x86: redis.cluster.xu1.l arge.r2.384 Arm: redis.cluster.au1.l arge.r2.384
512	512	64	640,000/3,200,000	49,152/49,152	500,000	x86: redis.cluster.xu1.l arge.r2.512 Arm: redis.cluster.au1.l arge.r2.512
768	768	96	960,000/4,800,000	73,728/73,728	500,000	x86: redis.cluster.xu1.l arge.r2.768 Arm: redis.cluster.au1.l arge.r2.768
1024	1024	128	1,280,000/6,400,000	98,304/98,304	500,000	x86: redis.cluster.xu1.l arge.r2.1024 Arm: redis.cluster.au1.l arge.r2.1024

## 1.5 Command Compatibility

## 1.5.1 Commands Supported and Disabled by DCS for Redis 3.0

DCS for Redis 3.0 is developed based on Redis 3.0.7 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 3.0's compatibility with Redis commands, including supported commands, disabled commands, unsupported scripts and commands of later Redis versions, and restrictions on command usage. For more information about the command syntax, visit the [Redis official website](#).

### NOTE

DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. DCS Redis 4.0 or 5.0 instances are recommended.

DCS for Redis instances support most Redis commands, which are listed in [Commands Supported by DCS for Redis 3.0](#). Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 3.0](#).
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Command Restrictions](#).
- Some Redis commands have usage restrictions, which are described in [Other Command Usage Restrictions](#).

## Commands Supported by DCS for Redis 3.0

The following lists commands supported by DCS for Redis 3.0.

### NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- The following commands listed in the tables are not supported by Proxy Cluster instances:
  - **List** group: **BLPOP**, **BRPOP**, and **BRPOPLRUSH**
  - **CLIENT** commands in the **Server** group: **CLIENT KILL**, **CLIENT GETNAME**, **CLIENT LIST**, **CLIENT SETNAME**, **CLIENT PAUSE**, and **CLIENT REPLY**.
  - **Server** group: **MONITOR**
  - **Key** group: **RANDOMKE** (for old Proxy Cluster instances)

**Table 1-15** Commands supported by DCS Redis 3.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE

Keys	String	Hash	List	Set	Sorted Set	Server
EXPIRE	BITPOS	HGET ALL	LINDEX	SDIFFST ORE	ZINCRBY	TIME
MOVE	DECR	HINC RBY	LINSER T	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINC RBYF LOAT	LLEN	SINTERS TORE	ZRANGEBYS CORE	KEYS
PTTL	GET	HKEY S	LPOP	SISMEM BER	ZRANK	CLIENT KILL
RANDOM KEY	GETRANG E	HMG ET	LPUSH X	SMEMBE RS	ZREMRANGE BYRANK	CLIENT LIST
RENAME	GETSET	HMSE T	LRANG E	SMOVE	ZREMRANGE BYCORE	CLIENT GETNAME
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTOR E	INCRBY	HSET NX	LSET	SRAND MEMBE R	ZREVRANGE BYSCORE	CONFIG GET
SORT	INCRBYFL OAT	HVAL S	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	-	RPOPL PU	SUNION STORE	ZUNIONSTO RE	ROLE
SCAN	MSETNX	-	RPOPL PUSH	SSCAN	ZINTERSTOR E	-
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	-
-	SET	-	RPUSH X	-	ZRANGEBYL EX	-
-	SETBIT	-	-	-	-	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANG E	-	-	-	-	-
-	STRLEN	-	-	-	-	-

**Table 1-16** Commands supported by DCS Redis 3.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

## Commands Disabled by DCS for Redis 3.0

The following lists commands disabled by DCS for Redis 3.0.

**Table 1-17** Redis commands disabled in single-node and master/standby DCS Redis 3.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF



**Table 1-18** Redis commands disabled in Proxy Cluster DCS Redis 3.0 instances

Keys	Server	List	Transactions	Connection	Cluster	codis
MIGRATE	SLAVEOF	BLPOP	DISCARD	SELECT	CLUSTER	TIME
MOVE	SHUTDOWN	BRPOP	EXEC	-	-	SLOTSINFO
-	LASTSAVE	BRPOPLPUSH	MULTI	-	-	SLOTSDEL
-	DEBUG commands	-	UNWATCH	-	-	SLOTSMGRSLOT
-	COMMAND	-	WATCH	-	-	SLOTSMGRTONE
-	SAVE	-	-	-	-	SLOTSCHECK
-	BGSAVE	-	-	-	-	SLOTSMGRRTAGSLOT
-	BGREWRITEAOF	-	-	-	-	SLOTSMGRRTAGONE
-	SYNC	-	-	-	-	-
-	PSYNC	-	-	-	-	-
-	MONITOR	-	-	-	-	-
-	CLIENT commands	-	-	-	-	-
-	OBJECT	-	-	-	-	-
-	ROLE	-	-	-	-	-

## 1.5.2 Commands Supported and Disabled by DCS for Redis 4.0

DCS for Redis 4.0 is developed based on Redis 4.0.14 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 4.0's compatibility with Redis commands, including supported and disabled commands. For more information about the command syntax, visit the [Redis official website](#).

DCS for Redis instances support most Redis commands, which are listed in [Commands Supported by DCS for Redis 4.0](#). Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 4.0](#).
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Command Restrictions](#).
- Some Redis commands have usage restrictions, which are described in [Other Command Usage Restrictions](#).

## Commands Supported by DCS for Redis 4.0

[Table 1-19](#) and [Table 1-20](#) list the Redis commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances.

[Table 1-21](#) and [Table 1-22](#) list the Redis commands supported by Proxy Cluster DCS Redis 4.0 instances.

### NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- For DCS Redis 4.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

**Table 1-19** Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST

Keys	String	Hash	List	Set	Sorted Set	Server
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGE BYSCORE	CLIENT GETNAME
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGE BYSCORE	CONFIG GET
SORT	INCRBY FLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPL PU	SUNION STORE	ZUNIONSTO RE	ROLE
SCAN	MSETNX	HLEN	RPOPL PUSH	SSCAN	ZINTERSTOR E	SWAPDB
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIRE	SET	-	RPUSH X	-	ZRANGEBYL EX	CONFIG
PEXPIRE AT	SETBIT	-	LPUSH	-	ZLEXCOUNT	COMMAN D
-	SETEX	-	-	-	ZREMRANGE BYSCORE	-
-	SETNX	-	-	-	ZREM	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

**Table 1-20** Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connecti on	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH

HyperLogLog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT (not supported by Redis Cluster instances)	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

**Table 1-21** Commands supported by Proxy Cluster DCS Redis 4.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	ROLE
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MEMORY
RENAME	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANKBYRANK	COMMAND
RENAME NX	GETSET	HMSET	LRANGE	SMOVE	ZREMRANKBYSCORE	COMMAND COUNT

Keys	String	Hash	List	Set	Sorted Set	Server
RESTORE	INCR	HSET	LREM	SPOP	ZREVRANGE	COMMAND GETKEYS
SORT	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	COMMAND INFO
TTL	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG GET
TYPE	MGET	HSCAN	RPOP	SUNION	ZSCORE	CONFIG RESETSTAT
SCAN	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	CONFIG REWRITE
OBJECT	MSETNX	HLEN	RPUSH	SSCAN	ZINTERSTORE	CONFIG SET
PEXPIRE	PSETEX	HKEYS	RPUSHX	-	ZSCAN	-
PEXPIREAT	SET	-	LPUSH	-	ZRANGEBYLEX	-
EXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZREVRANGEBYSCORE	-
TOUCH	SETNX	-	-	-	ZREM	-
UNLINK	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
-	STRLEN	-	-	-	ZREVRANGEBYLEX	-
-	BITFIELD	-	-	-	-	-
-	GETBIT	-	-	-	-	-

**Table 1-22** Commands supported by Proxy Cluster DCS Redis 4.0 instances (2)

HyperLog log	Pub/Sub	Transactions	Connect ion	Scripting	Geo	Cluster
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	CLUSTER INFO
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	CLUSTER NODES
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	CLUSTER SLOTS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	CLUSTER ADDSLOTS
-	SUBSCRIBE	WATCH	CLIENT KILL	SCRIPT KILL	GEORADIUS	ASKING
-	UNSUBSCRIBE	-	CLIENT LIST	SCRIPT LOAD	GEORADIUSBYMEMBER	READONLY
-	-	-	CLIENT GETNAME	SCRIPT DEBUG YES SYNC NO	GEOSEARCH	READWRITE
-	-	-	CLIENT SETNAME	-	GEOSEARCHSTORE	-

## Commands Disabled by DCS for Redis 4.0

The following lists commands disabled by DCS for Redis 4.0.

**Table 1-23** Redis commands disabled in single-node and master/standby DCS Redis 4.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	SAVE

Keys	Server
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

**Table 1-24** Redis commands disabled in Proxy Cluster DCS Redis 4.0 instances

Keys	Server	Sorted Set	Cluster
MIGRATE	BGREWRITEAOF	BZPOPMAX	READONLY
MOVE	BGSAVE	BZPOPMIN	READWRITE
RANDOMKEY	CLIENT commands	ZPOPMAX	-
WAIT	DEBUG OBJECT	ZPOPMIN	-
-	DEBUG SEGFAULT	-	-
-	LASTSAVE	-	-
-	PSYNC	-	-
-	SAVE	-	-
-	SHUTDOWN	-	-
-	SLAVEOF	-	-
-	LATENCY commands	-	-
-	MODULE commands	-	-
-	LOLWUT	-	-
-	SWAPDB	-	-
-	REPLICAOF	-	-
-	SYNC	-	-

**Table 1-25** Redis commands disabled in Redis Cluster DCS Redis 4.0 instances

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS

Keys	Server	Cluster
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

### 1.5.3 Commands Supported and Disabled by DCS for Redis 5.0

DCS for Redis 5.0 is developed based on Redis 5.0.14 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 5.0's compatibility with Redis commands, including supported and disabled commands. For more information about the command syntax, visit the [Redis official website](#).

DCS for Redis instances support most Redis commands. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 5.0](#).
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Command Restrictions](#).
- Some Redis commands have usage restrictions, which are described in [Other Command Usage Restrictions](#).

#### Commands Supported by DCS for Redis 5.0

- [Table 1-26](#) and [Table 1-27](#) list commands supported by single-node, master/standby, and Redis Cluster DCS for Redis 5.0.
- [Table 1-28](#) and [Table 1-29](#) list commands supported by Proxy Cluster DCS for Redis 5.0 instances.



 NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- For DCS Redis 5.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

**Table 1-26** Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 5.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGE	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT GETNAME
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB

Keys	String	Hash	List	Set	Sorted Set	Server
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIRE AT	SET	-	RPUSH X	-	ZRANGEBYLEX	CONFIG
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	COMMAND
-	SETEX	-	-	-	ZPOPMIN	-
-	SETNX	-	-	-	ZPOPMAX	-
-	SETRANGE	-	-	-	ZREMRANGEBYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	-	-

**Table 1-27** Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 5.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT (not supported by Redis Cluster instances)	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	-	-	-	XLEN
-	-	-	-	-	-	XPENDING

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
-	-	-	-	-	-	XRANGE
-	-	-	-	-	-	XREAD
-	-	-	-	-	-	XREADGROUP
-	-	-	-	-	-	XREVRANGE
-	-	-	-	-	-	XTRIM

**Table 1-28** Commands supported by Proxy Cluster DCS Redis 5.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	ROLE
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MEMORY
RENAME	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	COMMAND
RENAME NX	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	COMMAND COUNT
RESTORE	INCR	HSET	LREM	SPOP	ZREVRANGE	COMMAND GETKEYS
SORT	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	COMMAND INFO

Keys	String	Hash	List	Set	Sorted Set	Server
TTL	INCRBYF LOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG GET
TYPE	MGET	HSCAN	RPOP	SUNION	ZSCORE	CONFIG RESETSTAT
SCAN	MSET	HSTRLEN	RPOPLPUSH	SUNION STORE	ZUNION STORE	CONFIG REWRITE
OBJECT	MSETNX	HLEN	RPUSH	SSCAN	ZINTERSTORE	CONFIG SET
PEXPIRE	PSETEX	HKEYS	RPUSHX	-	ZSCAN	-
PEXPIREAT	SET	-	LPUSH	-	ZRANGEBYLEX	-
EXPIREAT	SETBIT	-	-	-	ZLEXCOUNT	-
KEYS	SETEX	-	-	-	ZREMRANGEBYSCORE	-
MIGRATE	SETNX	-	-	-	ZREM	-
UNLINK	SETRANGE	-	-	-	ZREMRANGEBYLEX	-
TOUCH	STRLEN	-	-	-	ZPOPMAX	-
-	BITFIELD	-	-	-	ZPOPMIN	-
-	GETBIT	-	-	-	BZPOPMAX	-
-	-	-	-	-	BZPOPMIN	-
-	-	-	-	-	ZREVRANGEBYLEX	-

**Table 1-29** Commands supported by Proxy Cluster DCS Redis 5.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connecti on	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	CLIENT KILL	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	CLIENT LIST	SCRIPT LOAD	GEORADIUSBY MEMBER
-	-	-	CLIENT GETNAME	SCRIPT DEBUG YES SYNC  NO	GEOSEARCH
-	-	-	CLIENT SETNAME	-	GEOSEARCHSTORE

## Commands Disabled by DCS for Redis 5.0

The following lists commands disabled by DCS for Redis 5.0.

**Table 1-30** Redis commands disabled in single-node and master/standby Redis 5.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC

Keys	Server
-	PSYNC

**Table 1-31** Redis commands disabled in Proxy Cluster DCS Redis 5.0 instances

Keys	Server	Cluster
MIGRATE	BGREWRITEAOF	READONLY
MOVE	BGSAVE	READWRITE
RANDOMKEY	CLIENT commands	-
WAIT	DEBUG OBJECT	-
-	DEBUG SEGFAULT	-
-	LASTSAVE	-
-	PSYNC	-
-	SAVE	-
-	SHUTDOWN	-
-	SLAVEOF	-
-	LATENCY commands	-
-	MODULE commands	-
-	LOLWUT	-
-	SWAPDB	-
-	REPLICAOF	-
-	SYNC	-

**Table 1-32** Redis commands disabled in Redis Cluster DCS Redis 5.0 instances

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH

Keys	Server	Cluster
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

## 1.5.4 Commands Supported and Disabled by DCS for Redis 6.0

DCS for Redis 6.0 is compatible with Redis 6.2.7 and with open-source protocols and commands.

This section describes DCS for Redis 6.0's command compatibility, including supported and disabled commands.

For more information about the command syntax, visit the [Redis official website](#).

DCS Redis instances support most Redis commands. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 6.0](#).
- Some Redis commands (such as **KEYS**, **FLUSHDB**, and **FLUSHALL**) have usage restrictions, which are described in [Other Command Usage Restrictions](#).

### Commands Supported by DCS for Redis 6.0

**Table 1-33** Commands supported by DCS for Redis 6.0 (1)

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	CONFIG GET
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MONITOR
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGE	SLOWLOG
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGE	ROLE
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	SWAPDB
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGE	MEMORY
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	ACL
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	COMMAND
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUSH	SMISMEMBER	ZSCAN	-
PEXPIREAT	SET	-	RPUSHX	-	ZRANGEBYLEX	-
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
KEYS	SETEX	-	BLMOVE	-	ZPOPMIN	-
COPY	SETNX	-	LMOVE	-	ZPOPMAX	-
-	SETRANGE	-	LPOS	-	ZREMRANGE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	ZDIFF	-



Generic (Key)	String	Hash	List	Set	Sorted Set	Server
-	BITFIELD_RO	-	-	-	ZDIFFSTORE	-
-	GETDEL	-	-	-	ZINTER	-
-	GETEX	-	-	-	ZMSCORE	-
-	-	-	-	-	ZRANDMEMBER	-
-	-	-	-	-	ZRANGESTORE	-
-	-	-	-	-	ZUNION	-

**Table 1-34** Commands supported by DCS for Redis 6.0 (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT (not supported by Redis Cluster instances)	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	CLIENT CACHING	SCRIPT LOAD	GEORADIUS BYMEMBER	XINFO
-	-	-	CLIENT GETREDIR	-	-	XLEN
-	-	-	CLIENT INFO	-	-	XPENDING

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
-	-	-	CLIENT TRACKING	-	-	XRANGE
-	-	-	CLIENT TRACKING INFO	-	-	XREAD
-	-	-	CLIENT UNPAUSE	-	-	XREADGROUP
-	-	-	CLIENT KILL	-	-	XREVRANGE
-	-	-	CLIENT LIST	-	-	XTRIM
-	-	-	CLIENT GETNAME	-	-	XAUTOCLAIM
-	-	-	CLIENT SETNAME	-	-	XGROUP CREATECONSUMER
-	-	-	HELLO	-	-	-
-	-	-	RESET	-	-	-

## Commands Disabled by DCS for Redis 6.0

**Table 1-35** Redis commands disabled in DCS Redis 6.0 instances

Generic (Key)	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET

Generic (Key)	Server	Cluster
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

### 1.5.5 Web CLI Commands

Web CLI is a command line tool provided on the DCS console. This section describes Web CLI's compatibility with Redis commands, including supported and disabled commands. For details about the command syntax, visit the [Redis official website](#).

**Currently, only DCS for Redis 4.0 and later support Web CLI.**

 **NOTE**

- Keys and values cannot contain spaces.
- If the value is empty, **nil** is returned after the **GET** command is executed.

### Commands Supported by Web CLI

The following lists the commands supported when you use Web CLI.

**Table 1-36** Commands supported by Web CLI (1)

Keys	String	List	Set	Sorted Set	Server
DEL	APPEND	R PUSH	SADD	ZADD	FLUSHALL
OBJECT	BITCOUNT	R PUSHX	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	LLEN	SINTERSTORE	ZRANGEBYSCORE	CLIENT KILL
PTTL	GET	LPOP	SISMEMBER	ZRANK	CLIENT LIST

Keys	String	List	Set	Sorted Set	Server
RANDOM KEY	GETRANGE	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT GETNAME
RENAME	GETSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT SETNAME
RENAMENX	INCR	LREM	SPOP	ZREVRANGE	CONFIG GET
SCAN	INCRBY	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	SLOWLOG
SORT	INCRBYFLOAT	LTRIM	SREM	ZREVRANK	ROLE
TTL	MGET	RPOP	SUNION	ZSCORE	SWAPDB
TYPE	MSET	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	MEMORY
-	MSETNX	RPOPLPUSH	SSCAN	ZINTERSTORE	-
-	PSETEX	-	-	ZSCAN	-
-	SET	-	-	ZRANGEBYLEX	-
-	SETBIT	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	-
-	SETNX	-	-	-	-
-	SETRANGE	-	-	-	-
-	STRLEN	-	-	-	-
-	BITFIELD	-	-	-	-

**Table 1-37** Commands supported by Web CLI (2)

Hash	HyperLog log	Connect ion	Scripting	Geo	Pub/Sub
HDEL	PFADD	AUTH	EVAL	GEOADD	UNSUBSCRIBE
HEXISTS	PFCOUNT	ECHO	EVALSHA	GEOHASH	PUBLISH
HGET	PFMERGE	PING	SCRIPT EXISTS	GEOPOS	PUBSUB

Hash	HyperLog log	Connect ion	Scripting	Geo	Pub/Sub
HGETALL	-	QUIT	SCRIPT FLUSH	GEODIST	PUNSUBSCRIBE
HINCRBY	-	-	SCRIPT KILL	GEORADIUS	-
HINCRBYFLOAT	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	-
HKEYS	-	-	-	-	-
HMGET	-	-	-	-	-
HMSET	-	-	-	-	-
HSET	-	-	-	-	-
HSETNX	-	-	-	-	-
HVALS	-	-	-	-	-
HSCAN	-	-	-	-	-
HSTRLEN	-	-	-	-	-

## Commands Disabled in Web CLI

The following lists the commands disabled when you use Web CLI.

**Table 1-38** Commands disabled in Web CLI (1)

Keys	Server	Transactions	Cluster
MIGRATE	SLAVEOF	UNWATCH	CLUSTER MEET
WAIT	SHUTDOWN	REPLICAOF	CLUSTER FLUSHSLOTS
DUMP	DEBUG commands	DISCARD	CLUSTER ADDSLOTS
RESTORE	CONFIG SET	EXEC	CLUSTER DELSLOTS
-	CONFIG REWRITE	MULTI	CLUSTER SETSLOT
-	CONFIG RESETSTAT	WATCH	CLUSTER BUMPEPOCH
-	SAVE	-	CLUSTER SAVECONFIG
-	BGSAVE	-	CLUSTER FORGET
-	BGREWRITEAOF	-	CLUSTER REPLICATE
-	COMMAND	-	CLUSTER COUNT-FAILURE-REPORTS

Keys	Server	Transactions	Cluster
-	KEYS	-	CLUSTER FAILOVER
-	MONITOR	-	CLUSTER SET-CONFIG-EPOCH
-	SYNC	-	CLUSTER RESET
-	PSYNC	-	-
-	ACL	-	-
-	MODULE	-	-

**Table 1-39** Commands disabled in Web CLI (2)

List	Connection	Sorted Set	Pub/Sub
BLPOP	SELECT	BZPOPMAX	PUBLISH
BRPOP	-	BZPOPMIN	SUBSCRIBE
BLMOVE	-	BZMPOP	-
BRPOPLPUSH	-	-	-
BLMPOP	-	-	-

## 1.5.6 Command Restrictions

Some Redis commands are supported by Redis Cluster DCS instances for multi-key operations in the same slot. For details, see [Table 1-40](#).

Some commands support multiple keys but do not support cross-slot access. For details, see [Table 1-41](#).

[Table 1-42](#) lists commands restricted for Proxy Cluster DCS Redis 4.0 instances.

**Table 1-40** Redis commands restricted in Redis Cluster DCS instances

Category	Description
<b>Set</b>	
SINTER	Returns the members of the set resulting from the intersection of all the given sets.
SINTERSTORE	Equal to <b>SINTER</b> , but instead of returning the result set, it is stored in <i>destination</i> .
SUNION	Returns the members of the set resulting from the union of all the given sets.

Category	Description
SUNIONSTORE	Equal to <b>SUNION</b> , but instead of returning the result set, it is stored in <i>destination</i> .
SDIFF	Returns the members of the set resulting from the difference between the first set and all the successive sets.
SDIFFSTORE	Equal to <b>SDIFF</b> , but instead of returning the result set, it is stored in <i>destination</i> .
SMOVE	Moves <b>member</b> from the set at <b>source</b> to the set at <i>destination</i> .
<b>Sorted Set</b>	
ZUNIONSTORE	Computes the union of <i>numkeys</i> sorted sets given by the specified keys.
ZINTERSTORE	Computes the intersection of <i>numkeys</i> sorted sets given by the specified keys.
<b>HyperLogLog</b>	
PFCOUNT	Returns the approximated cardinality computed by the HyperLogLog data structure stored at the specified variable.
PFMERGE	Merges multiple HyperLogLog values into a unique value.
<b>Keys</b>	
RENAME	Renames <i>key</i> to <i>newkey</i> .
RENAMENX	Renames <i>key</i> to <i>newkey</i> if <i>newkey</i> does not yet exist.
BITOP	Performs a bitwise operation between multiple keys (containing string values) and stores the result in the destination key.
RPOPLPUSH	Returns and removes the last element (tail) of the list stored at <i>source</i> , and pushes the element at the first element (head) of the list stored at <i>destination</i> .
<b>String</b>	
MSETNX	Merges multiple HyperLogLog values into a unique value.

 **NOTE**

While running commands that take a long time to run, such as **FLUSHALL**, DCS instances may not respond to other commands and may change to the faulty state. After the command finishes executing, the instance will return to normal.

**Table 1-41** Multi-key commands of Proxy Cluster instances

Category	Command
Multi-key commands that support cross-slot access	DEL, MGET, MSET, EXISTS, SUNION, SINTER, SDIFF, SUNIONSTORE, SINTERSTORE, SDIFFSTORE, ZUNIONSTORE, ZINTERSTORE
Multi-key commands that do not support cross-slot access	SMOVE, SORT, BITOP, MSETNX, RENAME, RENAMENX, BLPOP, BRPOP, RPOPLPUSH, BRPOPLPUSH, PFMERGE, PFCOUNT, BLMOVE, COPY, GEOSEARCHSTORE, LMOVE, ZRANGESTORE

**Table 1-42** Redis commands restricted for Proxy Cluster DCS Redis instances

Category	Command	Restriction
Set	SMOVE	For a Proxy Cluster instance, the source and destination keys must be in the same slot.
Sorted sets	BZPOPMAX	For a Proxy Cluster instance, all keys transferred must be in the same slot.
	BZPOPMIN	
Geo	GEORADIUS	<ul style="list-style-type: none"> <li>For a Proxy Cluster instance, all keys transferred must be in the same slot.</li> <li>For a Proxy Cluster instance with multiple databases, the <b>STORE</b> option is not supported.</li> </ul>
	GEORADIUSBYMEMBER	
	GEOSEARCHSTORE	
Connection	CLIENT KILL	<ul style="list-style-type: none"> <li>Only the following two formats are supported:                             <ul style="list-style-type: none"> <li>CLIENT KILL ip:port</li> <li>CLIENT KILL ADDR ip:port</li> </ul> </li> <li>The <b>id</b> field has a random value, and it does not meet the <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math> requirement.</li> </ul>
	CLIENT LIST	<ul style="list-style-type: none"> <li>Only the following two formats are supported:                             <ul style="list-style-type: none"> <li>CLIENT LIST</li> <li>CLIENT LIST [TYPE normal master replica pubsub]</li> </ul> </li> <li>The <b>id</b> field has a random value, and it does not meet the <math>idc1 &lt; idc2 \rightarrow Tc1 &lt; Tc2</math> requirement.</li> </ul>



Category	Command	Restriction
	SELECT index	<p>Multi-DB of Proxy Cluster instances can be implemented by changing the keys. This solution is not recommended.</p> <p>Constraints on supporting multi-DB for a Proxy Cluster instance:</p> <ol style="list-style-type: none"> <li>1. The backend storage rewrites keys based on certain rules. Keys in the exported RDB file are not the original keys but can still be accessed through the Redis protocol.</li> <li>2. The <b>FLUSHDB</b> command deletes keys one by one, which takes a long time.</li> <li>3. <b>SWAPDB</b> is not supported.</li> <li>4. The <b>INFO KEYSpace</b> command does not return data of multi-DB.</li> <li>5. The <b>DBSIZE</b> command is time-consuming. Do not use it in the code.</li> <li>6. If multi-DB is used, the performance of the <b>KEYS</b> and <b>SCAN</b> commands deteriorates by up to 50%.</li> <li>7. LUA scripts do not support multi-DB.</li> <li>8. The <b>RANDOMKEY</b> command does not support multi-DB.</li> <li>9. By default, multi-DB is disabled. Before enabling or disabling this option for an instance, clear the instance data.</li> </ol>
HyperLogLog	PFCOUNT	For a Proxy Cluster instance, all keys transferred must be in the same slot.
	PFMERGE	
Keys	RENAME	For a Proxy Cluster instance, all keys transferred must be in the same slot.
	RENAMENX	
	SCAN	Proxy Cluster instances do not support the <b>SCAN</b> command in pipelines.

Category	Command	Restriction
Lists	BLPOP	For a Proxy Cluster instance, all keys transferred must be in the same slot.
	BRPOP	
	BRPOPLPUSH	
Pub/Sub	PSUBSCRIBE	Proxy Cluster instances do not support keyspace event subscription, so there would be no keyspace event subscription failure.
Scripting	EVAL	<ul style="list-style-type: none"> <li>For a Proxy Cluster instance, all keys transferred must be in the same slot.</li> <li>When the multi-DB function is enabled for a Proxy Cluster instance, the <b>KEYS</b> parameter is modified. Pay attention to the <b>KEYS</b> parameter used in the Lua script.</li> </ul>
	EVASHA	

Category	Command	Restriction
Server	MEMORY DOCTOR	<p>For a Proxy Cluster instance, add the <i>ip:port</i> of the node at the end of the command.</p> <p>Do as follows to obtain the IP address and port number of a node (<b>MEMORY USAGE</b> is used as an example):</p> <ol style="list-style-type: none"> <li>1. Run the <b>cluster keyslot</b> <i>key</i> command to query the slot number of a key.</li> <li>2. Run the <b>icluster nodes</b> command to query the IP address and port number corresponding to the slot where the key is. If the required information is not returned after you run the <b>icluster nodes</b> command, your Proxy Cluster instance may be of an earlier version. In this case, run the <b>cluster nodes</b> command.</li> <li>3. Run the <b>MEMORY USAGE</b> <i>key ip:port</i> command. If multi-DB is enabled for the Proxy Cluster instance, run the <b>MEMORY USAGE</b> <i>xxx:As {key} ip:port</i> command, where <i>xxx</i> indicates the DB where the key value is. For example, DB0, DB1, and DB255 correspond to 000, 001, and 255, respectively. The following is an example for a single-DB Proxy Cluster instance: <pre> set key1 value1 OK get key1 value1 cluster keyslot key1 9189 icluster nodes xxx 192.168.00.00:1111@xxx xxx connected 10923-16383 xxx 192.168.00.01:2222@xxx xxx connected 0-5460 xxx 192.168.00.02:3333@xxx xxx connected 5461-10922 MEMORY USAGE key1 192.168.00.02:3333 54                     </pre> </li></ol>
	MEMORY HELP	
	MEMORY MALLOC-STATS	
	MEMORY PURGE	
	MEMORY STATS	
	MEMORY USAGE	
	MONITOR	

Category	Command	Restriction
Strings	BITOP	For a Proxy Cluster instance, all keys transferred must be in the same slot.
	MSETNX	
Transactions	WATCH	For a Proxy Cluster instance, all keys transferred must be in the same slot.
	MULTI	The order of cross-slot commands in a transaction is not guaranteed. The following commands cannot be used in transactions: <b>WATCH, MONITOR, RANDOMKEY, KEYS, SCAN, SUBSCRIBE, UNSUBSCRIBE, PSUBSCRIBE, PUNSUBSCRIBE, SCRIPT, EVAL, EVALSHA, DBSIZE, AUTH, FLUSHDB, FLUSHALL, CLIENT, MEMORY</b>
	EXEC	
Streams	XACK	Currently, Proxy Cluster instances do not support Streams.
	XADD	
	XCLAIM	
	XDEL	
	XGROUP	
	XINFO	
	XLEN	
	XPENDING	
	XRANGE	
	XTRIM	
	XREVRANGE	
	XREAD	
	XREADGROUP GROUP	
	XAUTOCLAIM	

## 1.5.7 Other Command Usage Restrictions

This section describes restrictions on some Redis commands.

### KEYS Command

In case of a large amount of cached data, running the **KEYS** command may block the execution of other commands for a long time or occupy exceptionally large

memory. Therefore, when running the **KEYS** command, describe the exact pattern and do not use fuzzy **keys \***. Do not use the **KEYS** command in the production environment. Otherwise, the service running will be affected.

## Commands in the Server Group

- While running commands that take a long time to run, such as **FLUSHALL**, DCS instances may not respond to other commands and may change to the faulty state. After the command finishes executing, the instance will return to normal.
- When the **FLUSHDB** or **FLUSHALL** command is run, execution of other service commands may be blocked for a long time in case of a large amount of cached data.

## EVAL and EVALSHA Commands

- When the **EVAL** or **EVALSHA** command is run, at least one key must be contained in the command parameter. Otherwise, the error message "ERR eval/evalsha numkeys must be bigger than zero in redis cluster mode" is displayed.
- When the **EVAL** or **EVALSHA** command is run, a cluster DCS Redis instance uses the first key to compute slots. Ensure that the keys to be operated in your code are in the same slot. For details, visit <https://redis.io/commands>.
- For the **EVAL** command:
  - You are advised to learn the Lua script features of Redis before running the **EVAL** command. For details, see <https://redis.io/commands/eval>.
  - The execution timeout time of a Lua script is 5 seconds. Time-consuming statements such as long-time sleep and large loop statements should be avoided.
  - When calling a Lua script, do not use random functions to specify keys. Otherwise, the execution results are inconsistent on the master and standby nodes.

## Debugging Lua Scripts

When you debug Lua scripts for Proxy Cluster and read/write splitting instances, only the asynchronous non-blocking mode **--ldb** is supported. The synchronous blocking mode **--ldb-sync-mode** is not supported. By default, the maximum concurrency on each proxy is **2**. This restriction does not apply to other instance types.

## Other Restrictions

- The time limit for executing a Redis command is 15 seconds. To prevent other services from failing, a master/replica switchover will be triggered after the command execution times out.

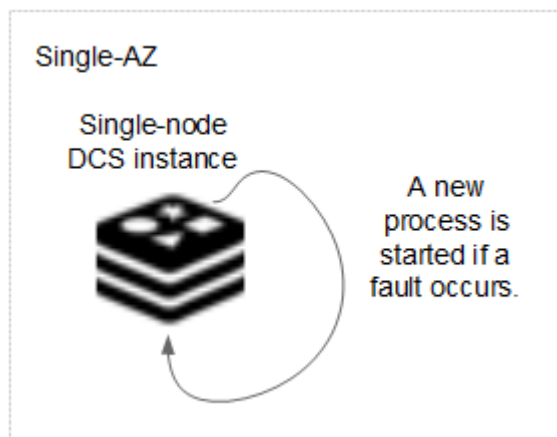
## 1.6 HA

### Single-AZ HA

Single-AZ deployment means to deploy an instance within a physical equipment room. DCS provides process/service HA, data persistence, and hot standby DR policies for different types of DCS instances.

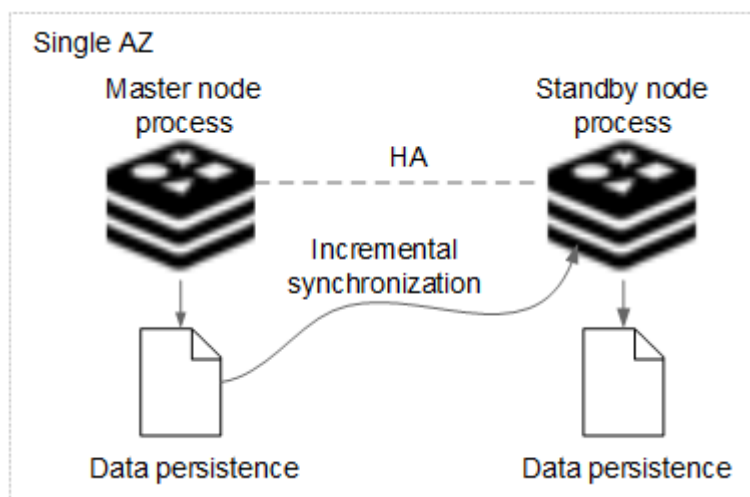
**Single-node DCS instance:** When DCS detects a process fault, a new process is started to ensure service HA.

**Figure 1-8** HA for a single-node DCS instance deployed within an AZ



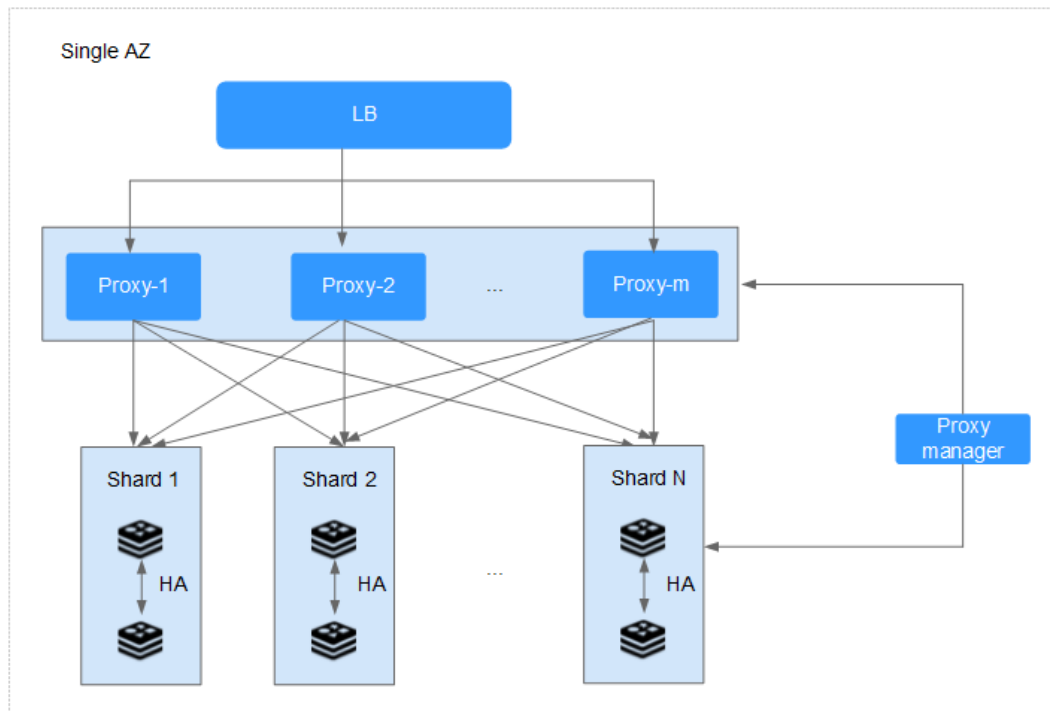
**Master/Standby DCS instance:** Data is persisted to disk in the master node and incrementally synchronized and persisted to the standby node, achieving hot standby and data persistence.

**Figure 1-9** HA for a master/standby DCS instance deployed within an AZ



**Cluster DCS instance:** Similar to a master/standby instance, data in each shard (instance process) of a cluster instance is synchronized between master and standby nodes and persisted on both nodes.

**Figure 1-10** HA for a cluster DCS instance deployed within an AZ



## 1.7 Comparing Redis Versions

When creating a DCS Redis instance, you can select the cache engine version and the instance type.

- **Version**

DCS supports Redis 3.0/4.0/5.0/6.0. The following table describes the differences between these versions.

**Table 1-43** Differences between Redis versions

Feature	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
Open-source compatibility	Redis 3.0.7	Redis 4.0.14 and 5.0.14, respectively	Redis 6.2.7
Instance deployment mode	Based on VMs	Containerized based on physical servers	Containerized based on physical servers

Feature	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds	8 seconds
QPS	50,000 QPS per node	50,000 QPS per node	100,000 QPS per node
Domain name access	Supported within a VPC	Supported within a VPC	Supported within a VPC
Visualized data management	Not supported	Web CLI for connecting to Redis and managing data	Web CLI for connecting to Redis and managing data
Instance type	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby, Proxy Cluster, , and Redis Cluster	Single-node, master/standby, Redis Cluster
Scale-up or scale-down	Online scale-up and scale-down	Online scale-up and scale-down	Online scale-up and scale-down
Backup and restoration	Supported for master/standby and cluster instances	Supported for master/standby and cluster instances	Supported for master/standby instances



 **NOTE**

The underlying architectures vary by Redis version. Once a Redis version is chosen, it cannot be changed. For example, you cannot upgrade a DCS Redis 3.0 instance to Redis 4.0 or 5.0. If you require a higher Redis version, create a new instance that meets your requirements and then migrate data from the old instance to the new one.

DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. You are advised to use DCS Redis 4.0 and later instances.

- **Instance type**

Select from single-node, master/standby, and cluster types. For details about their architectures and application scenarios, see [DCS Instance Types](#).

## 1.8 Comparing DCS and Open-Source Cache Services

DCS supports single-node, master/standby, and cluster instances, ensuring high read/write performance and fast data access. It also supports various instance management operations to facilitate your O&M. With DCS, you only need to focus on the service logic, without concerning about the deployment, monitoring, scaling, security, and fault recovery issues.

DCS is compatible with open-source Redis, and can be customized based on your requirements. This renders DCS unique features in addition to the advantages of open-source cache databases.

### DCS for Redis vs. Open-Source Redis

**Table 1-44** Differences between DCS for Redis and open-source Redis

Feature	Open-Source Redis	DCS for Redis
Service deployment	Requires 0.5 to 2 days to prepare servers.	<ul style="list-style-type: none"> <li>• Creates a Redis 3.0 instance in 5 to 15 minutes.</li> <li>• Creates a containerized Redis 4.0 or later instance within 8 seconds.</li> </ul>
Version	-	Deeply engaged in the open-source community and supports the latest Redis version. Redis 3.0, 4.0, 5.0, and 6.0 are supported.
Security	Network and server safety is the user's responsibility.	<ul style="list-style-type: none"> <li>• Network security is ensured using VPCs and security groups.</li> <li>• Data reliability is ensured by data replication and scheduled backup.</li> </ul>
Performance	-	50,000 QPS per node

Feature	Open-Source Redis	DCS for Redis
Monitoring	Provides only basic statistics.	<p>Provides more than 30 monitoring metrics and customizable alarm threshold and policies.</p> <ul style="list-style-type: none"> <li>● Various metrics                             <ul style="list-style-type: none"> <li>– External metrics include the number of commands, concurrent operations, connections, clients, and denied connections.</li> <li>– Resource usage metrics include CPU usage, physical memory usage, network input throughput, and network output throughput.</li> <li>– Internal metrics include instance capacity usage, as well as the number of keys, expired keys, PubSub channels, PubSub patterns, keyspaces hits, and keyspaces misses.</li> </ul> </li> <li>● Customize alarm thresholds and policies for different metrics to help identify service faults.</li> </ul>
Backup and restoration	Supported	<ul style="list-style-type: none"> <li>● Supports scheduled and manual backup. Backup files can be downloaded.</li> <li>● Backup data can be restored on the console.</li> </ul>
Parameter management	No visualized parameter management	<ul style="list-style-type: none"> <li>● Visualized parameter management is supported on the console.</li> <li>● Configuration parameters can be modified online.</li> <li>● Data can be accessed and modified on the console.</li> </ul>
Scale-up	Interrupts services and involves a complex procedure from modifying the server RAM to modifying Redis memory and restarting the OS and services.	<ul style="list-style-type: none"> <li>● Supports online scale-up and scale-down without interrupting services.</li> <li>● Specifications can be scaled up or down within the available range based on service requirements.</li> </ul>

## 1.9 Basic Concepts

### DCS Instance

An instance is the minimum resource unit provided by DCS.

DCS supports the Redis cache engine, and single-node, master/standby, and cluster instance types. For each instance type, multiple specifications are available.

For details, see [DCS Instance Specifications](#) and [DCS Instance Types](#).

### Project

Projects are used to group and isolate OpenStack resources (computing resources, storage resources, and network resources). A project can be a department or a project team. Multiple projects can be created for one account.

### Password-Free Access

DCS Redis instances can be accessed in the VPC without passwords. Latency is lower because no password authentication is involved.

You can enable password-free access for instances that do not have sensitive data.

### Maintenance Time Window

The maintenance time window is the period when the DCS service team upgrade and maintain the instance.

DCS instance maintenance takes place only once a quarter and does not interrupt services. Even so, you are advised to select a time period when the service demand is low.

When creating an instance, you must specify a maintenance time window, which can be modified on the **Basic Information** page after the instance is created.

### Cross-AZ Deployment

Master/Standby instances are deployed across different AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve HA for both data and applications.

When creating a master/standby or cluster DCS Redis instance, you can select a standby AZ for the node.

### Shard

A shard is a management unit of a cluster DCS Redis instance. Each shard corresponds to a redis-server process. A cluster consists of multiple shards. Each shard has multiple slots. Data is distributedly stored in the slots. The use of shards increases cache capacity and concurrent connections.

Each cluster instance consists of multiple shards. By default, each shard is a master/standby instance with two replicas. The number of shards is equal to the number of master nodes in a cluster instance.

## Replica

A replica is a node in a DCS instance. A single-replica instance has no standby node. A two-replica instance has one master node and one standby node. By default, each master/standby instance has two replicas. If the number of replicas is set to three for a master/standby instance, the instance has one master node and two standby nodes. A single-node instance has only one node.

## 1.10 Permissions Management

If you need to grant your enterprise personnel permission to access your DCS resources, use Identity and Access Management (IAM). IAM provides identity authentication, fine-grained permissions management, and access control. IAM helps you secure access to your cloud resources. If your cloud account does not require IAM for permissions management, you can skip this section.

IAM is a free service. You only pay for the resources in your account.

With IAM, you can control access to specific cloud resources. For example, if you want some software developers in your enterprise to be able to use DCS resources but do not want them to be able to delete DCS instances or perform any other high-risk operations, you can create IAM users and grant permission to use DCS instances but not permission to delete them.

IAM supports role/policy-based authorization and identity policy-based authorization.

The following table describes the differences between these two authorization models.

**Table 1-45** Differences between role/policy-based and identity policy-based authorization

Authorization Model	Core Relationship	Permissions	Authorization Method	Scenario
Role/Policy	User-permission-authorization scope	<ul style="list-style-type: none"> <li>System-defined roles</li> <li>System-defined policies</li> <li>Custom policies</li> </ul>	Assigning roles or policies to principals	To authorize a user, you need to add it to a user group first and then specify the scope of authorization. It provides a limited number of condition keys and cannot meet the requirements of fine-grained permissions control. This method is suitable for small- and medium-sized enterprises.
Identity policy	User-policy	<ul style="list-style-type: none"> <li>System-defined identity policies</li> <li>Custom identity policies</li> </ul>	<ul style="list-style-type: none"> <li>Assigning identity policies to principals</li> <li>Attaching identity policies to principals</li> </ul>	You can authorize a user by attaching an identity policy to it. User-specific authorization and a variety of key conditions allow for more fine-grained permissions control. However, this model can be hard to set up. It requires a certain amount of expertise and is suitable for medium- and large-sized enterprises.

Policies/identity policies and actions in the two authorization models are not interoperable. You are advised to use the identity policy-based authorization model. For details about system-defined permissions, see [Role/Policy-based Authorization](#) and [Identity Policy-based Authorization](#).

For more information about IAM, see *Identity and Access Management User Guide*.

## Role/Policy-based Authorization

DCS supports role/policy-based authorization. New IAM users do not have any permissions assigned by default. You need to first add them to one or more groups and then attach policies or roles to these groups. The users then inherit

permissions from the groups and can perform specified operations on cloud services based on the permissions they have been assigned.

DCS is a project-level service deployed for specific regions. When you set **Scope** to **Region-specific projects** and select the specified projects in the specified regions, the users only have permissions for DCS instances in the selected projects. If you set **Scope** to **All resources**, the users have permissions for DCS instances in all region-specific projects. When accessing DCS instances, the users need to switch to the authorized region.

**Table 1-46** lists all the system-defined permissions for DCS. System-defined policies in role/policy-based authorization are not interoperable with those in identity policy-based authorization.

**Table 1-46** System-defined permissions for DCS

Role/Policy Name	Description	Type	Dependencies
DCS FullAccess	All permissions for DCS. Users with these permissions can perform all operations on DCS instances.	System-defined policy	None
DCS UserAccess	Common user permissions for DCS, excluding permissions for creating, modifying, deleting DCS instances and modifying instance specifications.	System-defined policy	None
DCS ReadOnlyAccess	Read-only permissions for DCS. Users with these permissions can only view DCS data.	System-defined policy	None
DCS Administrator	Administrator permissions for DCS. Users with these permissions can operate and use all DCS instances.	System-defined role	<b>Server Administrator</b> and <b>Tenant Guest</b> roles, which must be attached in the same project as the <b>DCS Administrator</b> role

**Table 1-47** lists the common operations supported by system-defined permissions for DCS.

**Table 1-47** Common operations supported by system-defined permissions

Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess	DCS Administrator
Modifying instance configuration parameters	√	√	×	√
Deleting background tasks	√	√	×	√
Web CLI	√	√	×	√
Modifying instance running status	√	√	×	√
Expanding instance capacity	√	×	×	√
Changing instance passwords	√	√	×	√
Modifying DCS instances	√	×	×	√
Performing a master/standby switchover	√	√	×	√
Backing up instance data	√	√	×	√
Analyzing big keys or hot keys	√	√	×	√
Creating DCS instances	√	×	×	√
Deleting instance backup files	√	√	×	√

Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess	DCS Administrator
Restoring instance data	√	√	×	√
Resetting instance passwords	√	√	×	√
Migrating instance data	√	√	×	√
Downloading instance backup data	√	√	×	√
Deleting DCS instances	√	×	×	√
Querying instance configuration parameters	√	√	√	√
Querying instance restoration logs	√	√	√	√
Querying instance backup logs	√	√	√	√
Querying DCS instances	√	√	√	√
Querying instance background tasks	√	√	√	√
Querying all instances	√	√	√	√



Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess	DCS Administrator
Operating slow queries	√	√	√	√

## Identity Policy-based Authorization

DCS supports identity policy-based authorization. [Table 1-48](#) lists all the system-defined identity policies for DCS. System-defined policies in identity policy-based authorization are not interoperable with those in role/policy-based authorization.

**Table 1-48** System-defined identity policies for DCS

Identity Policy Name	Description	Type
DCSServiceLinkedAgencyPolicy	Agency permissions required by DCS for migrating faulty instances. No other operations are involved.	System-defined identity policy
DCSReadOnlyAccessPolicy	Read-only permissions for DCS.	System-defined identity policy
DCSUserAccessPolicy	Common user permissions for DCS, excluding permissions for creating, modifying, deleting DCS instances and modifying instance specifications.	System-defined identity policy
DCSFullAccessPolicy	Full permissions for DCS.	System-defined identity policy

[Table 1-49](#) lists the common operations supported by system-defined identity policies for DCS.

**Table 1-49** Common operations supported by system-defined policies

Operation	DCSReadOnlyAccessPolicy	DCSUserAccessPolicy	DCSFullAccessPolicy
Modifying instance configuration parameters	×	√	√

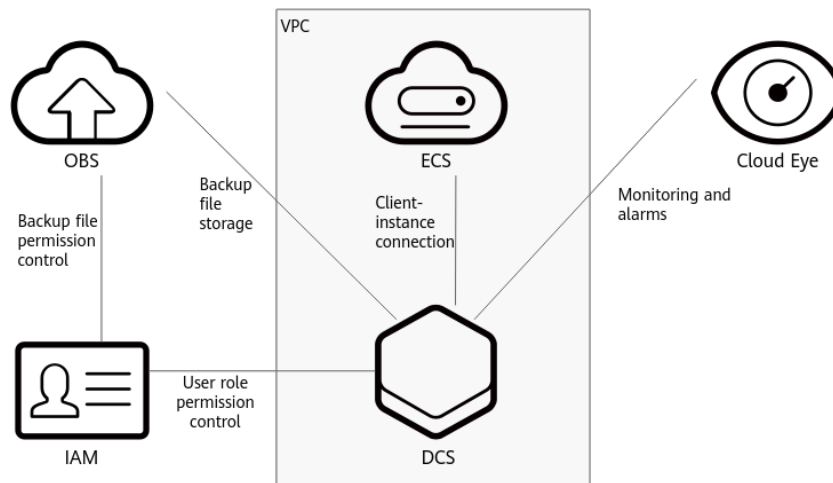
Operation	DCSReadOnlyAccess-Policy	DCSUserAccessPo- licy	DCSFullAccessP olicy
Deleting background tasks	×	√	√
Web CLI	×	√	√
Modifying instance running status	×	√	√
Expanding instance capacity	×	×	√
Changing instance passwords	×	√	√
Modifying DCS instances	×	×	√
Performing a master/standby switchover	×	√	√
Backing up instance data	×	√	√
Analyzing big keys or hot keys	×	√	√
Creating DCS instances	×	×	√
Deleting instance backup files	×	√	√
Restoring instance data	×	√	√
Resetting instance passwords	×	√	√
Migrating instance data	×	√	√
Downloading instance backup data	×	√	√

Operation	DCSReadOnlyAccess-Policy	DCSUserAccessPo- licy	DCSFullAccessP olicy
Deleting DCS instances	×	×	√
Querying instance configuration parameters	√	√	√
Querying instance restoration logs	√	√	√
Querying instance backup logs	√	√	√
Querying DCS instances	√	√	√
Querying instance background tasks	√	√	√
Querying all instances	√	√	√
Operating slow queries	√	√	√

## 1.11 Related Services

DCS is used together with other services, including VPC, ECS, IAM, Cloud Eye, and Object Storage Service (OBS).

**Figure 1-11** Relationships between DCS and other services



## VPC

A VPC is an isolated virtual network environment on the cloud. You can configure IP address ranges, subnets, and security groups in a VPC.

DCS runs in VPCs. The VPC service manages EIPs and bandwidth, and provides security groups. You can configure access rules for security groups to secure the access to DCS.

## ECS

An ECS is a cloud server that provides scalable, on-demand computing resources for secure, flexible, and efficient applications.

You can access and manage your DCS instances using an ECS.

## IAM

IAM provides identity authentication, permissions management, and access control.

With IAM, you can control access to DCS.

## Cloud Eye

Cloud Eye is a secure, scalable, and integrated monitoring service. With Cloud Eye, you can monitor your DCS service and configure alarm rules and notifications.

## OBS

OBS provides secure, cost-effective storage service using objects as storage units. With OBS, you can store and manage the lifecycle of massive amounts of data.

You can store DCS instance backup files in OBS.

# 2 Getting Started

---

## 2.1 Creating an Instance

### 2.1.1 Identifying Requirements

Before creating a DCS instance, identify your requirements and complete the following preparations:

1. Decide on the required cache engine version.  
Different Redis versions have different features. For details, see [Comparing Redis Versions](#).
2. Decide on the required instance type.  
DCS provides single-node, master/standby, Proxy Cluster, and Redis Cluster types of instances. Each type has its own architecture. For details about the instance architectures, see [DCS Instance Types](#).
3. Decide on the required instance specification.  
Each specification specifies the maximum available memory, number of connections, and bandwidth. For details, see [DCS Instance Specifications](#).
4. Decide whether backup policies are required.  
Currently, backup restoration policies can be configured for instances other than single-node ones. For details about backup and restoration, see [Overview](#).


### 2.1.2 Preparing Required Resources

To access DCS instances through a Virtual Private Cloud (VPC), create a VPC and configure security groups and subnets for it before using DCS. A VPC provides an isolated virtual network environment which you can configure and manage. Using VPCs enhances cloud resource security and simplifies network deployment.

Once you have created the required resources, you can use them for all DCS instances you subsequently create.

## Creating a VPC and Subnet

**Step 1** Log in to the management console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** Click **Service List**, and choose **Network > Virtual Private Cloud** to launch the VPC console.

**Step 4** Click **Apply for VPC**.

**Step 5** Create a VPC as prompted, retaining the default values unless otherwise required.

For details about how to create a VPC, see "VPC and Subnet" > "VPC" > "Creating a VPC" in *Virtual Private Cloud User Guide*.

After a VPC is created, a subnet is also created in the subnet. If the VPC needs more subnets, go to [Step 6](#) and [Step 7](#). Otherwise, go to [Creating a Security Group](#).

### NOTE

- When creating a VPC, **CIDR Block** indicates the IP address range of the VPC. If this parameter is set, the IP addresses of subnets in the VPC must be within the IP address range of the VPC.
- If you create a VPC to provision DCS instances, you do not need to configure the CIDR block for the VPC.

**Step 6** In the navigation pane on the left, choose **Subnets**.

**Step 7** Click **Create Subnet**. Create a subnet as prompted, retaining the default values unless otherwise required.

For details about how to create a subnet, see "VPC and Subnet" > "Subnet" in *Virtual Private Cloud User Guide*.

----End

## Creating a Security Group

### NOTE

Only DCS Redis 3.0 instances require security groups.

**Step 1** Log in to the VPC console.

**Step 2** In the navigation pane on the left, choose **Access Control > Security Groups** and then click **Create Security Group** in the upper right corner of the displayed page. Create a security group as prompted, retaining the default values unless otherwise required.

For details about how to create a security group, see "Security" > "Security Group" > "Creating a Security Group" in *Virtual Private Cloud User Guide*.

----End

## 2.1.3 Creating a DCS Redis Instance

You can create one or more DCS Redis instances with the required computing capabilities and storage space based on service requirements.

### NOTE


- DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. You are advised to use DCS Redis 4.0 and later instances.
- You cannot upgrade the Redis version of an instance. For example, a single-node DCS instance cannot be upgraded from Redis 4.0 to Redis 5.0. If you need Redis features of later versions, create a DCS Redis instance of a later version and then migrate data from the earlier instance to the new one.
- Single-node instances cannot ensure data persistence and do not support manual or scheduled data backup. Exercise caution before using them.

### Prerequisites

You have prepared [necessary resources](#).

### Creating a DCS Redis Instance

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner of the console and select a region and a project.

**Step 3** Click **Create DCS Instance**.

**Step 4** Select a region closest to your application to reduce latency and accelerate access.

**Step 5** Specify the following instance parameters based on the information collected in [Identifying Requirements](#).

1. **Cache Engine:**  
Select **Redis**.
2. **Version:**  
Currently, versions 4.0/5.0/6.0 are supported.
3. Set **Instance Type** to **Single-node, Master/Standby, Proxy Cluster** or **Redis Cluster**.
4. Select a CPU architecture.
5. Set **Replicas**. The default value is **2** (including the master).  
This parameter is displayed only when you select Redis 4.0 or later and the instance type is master/standby or Redis Cluster.
6. Select an AZ.

### NOTE

- To accelerate access, deploy your instance and your application in the same AZ.
  - There are multiple AZs in each region. If resources are insufficient in an AZ, the AZ will be unavailable. In this case, select another AZ.
7. **Instance Specification:**  
The remaining quota is displayed on the console.  
To apply to increase quota, click **Increase quota** below the specifications.

**Step 6** Configure the instance network parameters.

1. Select a VPC and a subnet.
2. Configure the IP address.

Redis Cluster instances only support automatically-assigned IP addresses. The other instance types support both automatically-assigned IP addresses and manually-specified IP addresses. You can manually specify an IP address available in your subnet.

For a DCS Redis 4.0 or later instance, you can specify a port number in the range from 1 to 65535. If no port is specified, the default port 6379 will be used. For a DCS Redis 3.0 instance, the port cannot be customized. Port 6379 will be used.

3. Select a security group.

A security group is a set of rules that control access to ECSs. It provides access policies for mutually trusted ECSs with the same security protection requirements in the same VPC.

This parameter is displayed only for DCS Redis 3.0 instances. DCS Redis 4.0/5.0/6.0 instances are based on VPC endpoints and do not support security groups.

**Step 7** Set **Name**.

The value of **Name** contains at least 4 characters. When you create multiple instances at a time, the instances are named in the format of *custom name-n*, where *n* starts from 000 and is incremented by 1. For example, if you create two instances and set **name** to **dc\_demo**, the two instances are respectively named as **dc\_demo-000** and **dc\_demo-001**.

**Step 8** Set the instance password.

- Select **Yes** or **No** for **Password Protected**.

 **NOTE**

- Password-free access carries security risks. Exercise caution when selecting this mode.
- After creating a DCS Redis instance to be accessed in password-free mode, you can set a password for it by using the password reset function. For details, see [Changing Password Settings for DCS Redis Instances](#).
- **Password** and **Confirm Password**: These parameters indicate the password of accessing the DCS Redis instance, and are displayed only when **Password Protected** is set to **Yes**.

 **NOTE**

For security purposes, if password-free access is disabled, the system prompts you to enter an instance-specific password when you are accessing the DCS Redis instance. Keep your instance password secure and change it periodically.

**Step 9** Choose whether to enable **Auto Backup**.

This parameter is displayed only when the instance type is master/standby or cluster. For more information on how to configure a backup policy, see [Overview](#).

**Step 10** Specify the number of instances to create.



**Step 11** Click **More Settings** to configure more parameters.

1. Enter a description of the instance.
2. Rename critical commands.

**Command Renaming** is displayed for Redis 4.0 and later. Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, **HGETALL**, **SCAN**, **HSCAN**, **SSCAN**, and **ZSCAN** commands. For Proxy Cluster instances, you can also rename the **DBSIZE** and **DBSTATS** commands.

3. Specify the maintenance window.

Choose a window for DCS O&M personnel to perform maintenance on your instance. You will be contacted before any maintenance activities are performed.

**Step 12** Click **Create Now**.

The displayed page shows the instance information you have specified.

**Step 13** Confirm the instance information and click **Submit**.

**Step 14** Return to the **Cache Manager** page to view and manage your DCS instances.

1. DCS Redis 4.0 and later instances are containerized and can be created within seconds.
2. After a DCS instance has been successfully created, it enters the **Running** state by default.

----End

## 2.2 Accessing an Instance

### 2.2.1 Network Conditions for Accessing DCS Redis

You can access a DCS instance through any Redis client. For details about Redis clients, see the [Redis official website](#).

There are different constraints when a client connects to Redis in certain ways:

- Accessing a Redis instance on a client within the same VPC  
The ECS where the client is installed must be in the same VPC as the DCS Redis instance. An ECS and a DCS instance can communicate with each other only when they belong to the same VPC. **Redis 3.0:** The instance and the ECS must either be configured with the same security group or use different security groups but can communicate with each other as configured by the security group rules. **Redis 4.0/5.0/6.0:** The IP address of the ECS must be on the whitelist of the DCS instance.
- Accessing a Redis instance on a client across VPCs in the same region  
If the client and DCS Redis instance are not in the same VPC, connect them by establishing a VPC peering connection. For details, see "Does DCS Support Cross-VPC Access?" in *Distributed Cache Service User Guide* > FAQs.
- Accessing a Redis instance of another region on a client  
If the client server and the Redis instance are not in the same region, connect the network using Direct Connect. For details, see *Direct Connect User Guide*.

To access a Redis instance across regions, the instance domain names cannot be resolved across regions. Therefore, the instance cannot be accessed at its domain name addresses. You can manually map the domain name to the IP address in the **hosts** file, or access the instance at its IP address.

Others:

Ensure that the security group and network ACL of the Redis client allow inbound access to the Redis server. Otherwise, Redis connections may become unstable.

## 2.2.2 Accessing a DCS Redis Instance Through `redis-cli`

Access a DCS Redis instance through `redis-cli` on an ECS in the same VPC. For more information on how to use other Redis clients, visit <https://redis.io/clients>.

### NOTE

- Redis 3.0 does not support port customization and allows only port 6379. For Redis 4.0 and later, you can specify a port or use the default port 6379. The following uses the default port 6379. If you have specified a port, replace 6379 with the actual port.
- **When connecting to a Redis Cluster instance, ensure that `-c` is added to the command.** Otherwise, the connection will fail.
  - Run the following command to connect to a Redis Cluster instance:  

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c
```
  - Run the following command to connect to a single-node, master/standby, or Proxy Cluster instance:  

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password}
```

For details, see [Step 3](#) and [Step 4](#).

## Prerequisites

- The DCS Redis instance you want to access is in the **Running** state.
- An ECS has been created. For more information on how to create ECSs, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS. Run the following command to install GCC if needed:

```
yum install -y make
yum install -y pcre-devel
yum install -y zlib-devel
yum install -y libevent-devel
yum install -y openssl-devel
yum install -y gcc-c++
```

## Procedure (Linux)

**Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

**Step 2** Install `redis-cli`.

The following steps assume that your client is installed on the Linux OS.

1. Log in to the ECS.
2. Run the following command to download the source code package of your Redis client from <https://download.redis.io/releases/redis-6.2.13.tar.gz>:

```
wget http://download.redis.io/releases/redis-6.2.13.tar.gz
```

 NOTE

The following uses redis-6.2.13 as an example. For details, see the [Redis official website](#).

3. Run the following command to decompress the source code package of your Redis client:

```
tar -xzf redis-6.2.13.tar.gz
```

4. Run the following commands to go to the Redis directory and compile the source code of your Redis client:

```
cd redis-6.2.13
```

```
make
```

```
cd src
```




**Step 3** Access a DCS instance of a type other than Redis Cluster.

Perform the following procedure to access a single-node, master/standby, or Proxy Cluster instance.

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password}
```

 NOTE

1. If the instance is password-free, connect it by running the `./redis-cli -h {dcs_instance_address} -p 6379` command.
2. If the instance is password-protected, connect it by running the `./redis-cli -h {dcs_instance_address} -p 6379 -a {password}` command.
3. If you forget the instance password or need to reset the password, choose **More > Reset Password** in the **Operation** column of the instance on the instance list.
4. *{dcs\_instance\_address}* can be the **Connection Address** (domain name) or **IP Address**. For details, see the *Distributed Cache Service User Guide > "FAQs" > "Client and Network Connection" > "Should I Use Domain Name or IP Address to Connect to a Redis Instance?"*.

Connection 	
Password Protected	Yes
Connection Address	redis-dd <span style="background-color: #e0e0e0; padding: 0 20px;"></span> n:6379 
IP Address	10.2...08:6379 

**Step 4** Access a DCS instance of the Redis Cluster type.

Do as follows to access a Redis Cluster instance:

1. Run the following commands to access the chosen DCS Redis instance:

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c
```

*{dcs\_instance\_address}* indicates the IP address/domain name of the DCS Redis instance, **6379** is the port used for accessing the instance, *{password}* is

the password of the instance, and `-c` is used for accessing Redis Cluster nodes. The IP address/domain name and port number are obtained in [Step 1](#).

Example:

```
root@ecs-redis:~/redis-6.2.13/src# ./redis-cli -h 192.168.0.85 -p 6379 -a ***** -c
192.168.0.85:6379>
```

2. Run the following command to view the Redis Cluster node information:

### cluster nodes

Each shard in a Redis Cluster has a master and a replica by default. The proceeding command provides all the information of cluster nodes.

```
192.168.0.85:6379> cluster nodes
0988ae8fd3686074c9afdce73d7878c81a33ddc 192.168.0.231:6379@16379 slave
f0141816260ca5029c56333095f015c7a058f113 0 1568084030
000 3 connected
1a32d809c0b743bd83b5e1c277d5d201d0140b75 192.168.0.85:6379@16379 myself,master - 0
1568084030000 2 connected 5461-10922
c8ad7af9a12cce3c8e416fb67bd6ec9207f0082d 192.168.0.130:6379@16379 slave
1a32d809c0b743bd83b5e1c277d5d201d0140b75 0 1568084031
000 2 connected
7ca218299c254b5da939f8e60a940ac8171adc27 192.168.0.22:6379@16379 master - 0 1568084030000
1 connected 0-5460
f0141816260ca5029c56333095f015c7a058f113 192.168.0.170:6379@16379 master - 0
1568084031992 3 connected 10923-16383
19b1a400815396c6223963b013ec934a657bdc52 192.168.0.161:6379@16379 slave
7ca218299c254b5da939f8e60a940ac8171adc27 0 1568084031
000 1 connected
```

Write operations can only be performed on master nodes. The CRC16 of the key modulo 16384 is taken to compute what is the hash slot of a given key.

As shown in the following, the value of **CRC16 (KEY) mode 16384** determines the hash slot that a given key is located at and redirects the client to the node where the hash slot is located at.

```
192.168.0.170:6379> set hello world
-> Redirected to slot [866] located at 192.168.0.22:6379
OK
192.168.0.22:6379> set happy day
OK
192.168.0.22:6379> set abc 123
-> Redirected to slot [7638] located at 192.168.0.85:6379
OK
192.168.0.85:6379> get hello
-> Redirected to slot [866] located at 192.168.0.22:6379
"world"
192.168.0.22:6379> get abc
-> Redirected to slot [7638] located at 192.168.0.85:6379
"123"
192.168.0.85:6379>
```

----End

## Procedure (Windows)

**Download** the compilation package of the Redis client for Windows. (This is not the source code package.) Decompress the package in any directory, open the CLI tool `cmd.exe`, and go to the directory. Then, run the following command to access the DCS Redis instance:

```
redis-cli.exe -h XXX -p 6379
```

**XXX** indicates the IP address/domain name of the DCS instance and **6379** is an example port number used for accessing the DCS instance. For details about how

to obtain the IP address/domain name and port number, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance". Change the address and port as required.

## 2.2.3 Access in Different Languages

### 2.2.3.1 Java

#### 2.2.3.1.1 Connecting to Redis on Jedis (Java)

This section describes how to access a Redis instance on Jedis. For more information about how to use other Redis clients, visit [the Redis official website](#).

Spring Data Redis is already integrated with [Jedis](#) and [Lettuce](#) for Spring Boot projects. Spring Boot 1.x is integrated with Jedis, and Spring Boot 2.x is integrated with Lettuce. To use Jedis in Spring Boot 2.x and later, you need to solve Lettuce dependency conflicts.

#### NOTE

Springboot 2.3.12.RELEASE or later is required. Jedis [3.10.0](#) or later is required.

### Prerequisites

- A Redis instance is created, and is in the **Running** state.
- View the IP address/domain name and port number of the DCS Redis instance to be accessed. For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

### Pom Configuration

```
<!-- import spring-data-redis -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
<!--In Spring Boot 2.0, Lettuce is used by default. To use Jedis, solve dependency conflicts.-->
  <exclusions>
    <exclusion>
      <groupId>io.lettuce</groupId>
      <artifactId>lettuce-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<!--Jedis dependency>
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>${jedis.version}</version>
</dependency>
```

### application.properties Configuration

- Single-node, master/standby, and Proxy Cluster

```
#Redis host
spring.redis.host=<host>
#Redis port
spring.redis.port=<port>
#Redis database number
spring.redis.database=0
```

```
#Redis password
spring.redis.password=<password>
#Redis read/write timeout
spring.redis.timeout=2000
#Whether to enable connection pooling
spring.redis.jedis.pool.enabled=true
#Minimum connections in the pool
spring.redis.jedis.pool.min-idle=50
#Maximum idle connections in the pool
spring.redis.jedis.pool.max-idle=200
#Maximum connections in the pool
spring.redis.jedis.pool.max-active=200
#Maximum amount of time a connection allocation should block before throwing an exception when
the pool is exhausted. The default value -1 indicates to wait indefinitely.
spring.redis.jedis.pool.max-wait=3000
#Interval for checking and evicting idle connection. Default: 60s.
spring.redis.jedis.pool.time-between-eviction-runs=60S
```

- **Redis Cluster**

```
#Redis Cluster node connection information
spring.redis.cluster.nodes=<ip:port>,<ip:port>,<ip:port>
#Redis Cluster password
spring.redis.password=<password>
#Redis Cluster max. redirecting times
spring.redis.cluster.max-redirects=3
#Redis read/write timeout
spring.redis.timeout=2000
#Whether to enable connection pooling
spring.redis.jedis.pool.enabled=true
#Minimum connections in the pool
spring.redis.jedis.pool.min-idle=50
#Maximum idle connections in the pool
spring.redis.jedis.pool.max-idle=200
#Maximum connections in the pool
spring.redis.jedis.pool.max-active=200
#Maximum amount of time a connection allocation should block before throwing an exception when
the pool is exhausted. The default value -1 indicates to wait indefinitely.
spring.redis.jedis.pool.max-wait=3000
#Interval for checking and evicting idle connections. Default: 60s.
spring.redis.jedis.pool.time-between-eviction-runs=60S
```

## Bean Configuration

- **Single-node, master/standby, and Proxy Cluster**

```
import java.time.Duration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.jedis.JedisClientConfiguration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;

import redis.clients.jedis.JedisPoolConfig;

@Configuration
public class RedisConfiguration {

    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password:}")
```

```
private String redisPassword;

@Value("${redis.connect.timeout:3000}")
private Integer redisConnectTimeout = 3000;

@Value("${redis.read.timeout:2000}")
private Integer redisReadTimeout = 2000;

@Value("${redis.pool.minSize:50}")
private Integer redisPoolMinSize = 50;

@Value("${redis.pool.maxSize:200}")
private Integer redisPoolMaxSize = 200;

@Value("${redis.pool.maxWaitMillis:3000}")
private Integer redisPoolMaxWaitMillis = 3000;

@Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

@Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

@Bean
public RedisConnectionFactory redisConnectionFactory(JedisClientConfiguration
clientConfiguration) {

    RedisStandaloneConfiguration standaloneConfiguration = new RedisStandaloneConfiguration();
    standaloneConfiguration.setHostName(redisHost);
    standaloneConfiguration.setPort(redisPort);
    standaloneConfiguration.setDatabase(redisDatabase);
    standaloneConfiguration.setPassword(redisPassword);

    return new JedisConnectionFactory(standaloneConfiguration, clientConfiguration);
}

@Bean
public JedisClientConfiguration clientConfiguration() {

    JedisClientConfiguration clientConfiguration = JedisClientConfiguration.builder()
        .connectTimeout(Duration.ofMillis(redisConnectTimeout))
        .readTimeout(Duration.ofMillis(redisReadTimeout))
        .usePooling().poolConfig(redisPoolConfig())
        .build();

    return clientConfiguration;
}

private JedisPoolConfig redisPoolConfig() {

    JedisPoolConfig poolConfig = new JedisPoolConfig();
    //Minimum connections in the pool
    poolConfig.setMinIdle(redisPoolMinSize);
    //Maximum idle connections in the pool
    poolConfig.setMaxIdle(redisPoolMaxSize);
    //Maximum total connections in the pool
    poolConfig.setMaxTotal(redisPoolMaxSize);
    //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
    poolConfig.setBlockWhenExhausted(true);
    //Longest time to wait for connection after pool is exhausted. The default value -1 indicates to
    wait indefinitely.
    poolConfig.setMaxWaitMillis(redisPoolMaxWaitMillis);
    //Set to true to enable connectivity test on creating connections. Default: false.
    poolConfig.setTestOnCreate(false);
    //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
    heavy-traffic services to reduce overhead.
    poolConfig.setTestOnBorrow(true);
    //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
    heavy-traffic services to reduce overhead.
```

```
poolConfig.setTestOnReturn(false);
//Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
poolConfig.setTestWhileIdle(true);
//Duration after which idle connections are evicted. If the idle duration is greater than this value
and the maximum number of idle connections is reached, idle connections are directly evicted.
poolConfig.setSoftMinEvictableIdleTimeMillis(redisPoolSoftMinEvictableIdleTimeMillis);
//Disable MinEvictableIdleTimeMillis().
poolConfig.setMinEvictableIdleTimeMillis(-1);
//Interval for checking and evicting idle connections. Default: 60s.
poolConfig.setTimeBetweenEvictionRunsMillis(redisPoolBetweenEvictionRunsMillis);
return poolConfig;
}
}
```

- **Redis Cluster**

```
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.jedis.JedisClientConfiguration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;

import redis.clients.jedis.JedisPoolConfig;

@Configuration
public class RedisConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:3000}")
    private Integer redisPoolMaxWaitMillis = 3000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(JedisClientConfiguration
clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
```



```
        clusterNodes.add(new RedisNode(nodeInfo[0], Integer.valueOf(nodeInfo[1])));
    }
    clusterConfiguration.setClusterNodes(clusterNodes);

    clusterConfiguration.setPassword(redisPassword);
    clusterConfiguration.setMaxRedirects(3);

    return new JedisConnectionFactory(clusterConfiguration, clientConfiguration);
}

@Bean
public JedisClientConfiguration clientConfiguration() {

    JedisClientConfiguration clientConfiguration = JedisClientConfiguration.builder()
        .connectTimeout(Duration.ofMillis(redisConnectTimeout))
        .readTimeout(Duration.ofMillis(redisReadTimeout))
        .usePooling().poolConfig(redisPoolConfig())
        .build();

    return clientConfiguration;
}

private JedisPoolConfig redisPoolConfig() {

    JedisPoolConfig poolConfig = new JedisPoolConfig();
    //Minimum connections in the pool
    poolConfig.setMinIdle(redisPoolMinSize);
    //Maximum idle connections in the pool
    poolConfig.setMaxIdle(redisPoolMaxSize);
    //Maximum total connections in the pool
    poolConfig.setMaxTotal(redisPoolMaxSize);
    //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
    poolConfig.setBlockWhenExhausted(true);
    //Longest time to wait for connection after pool is exhausted. The default value -1 indicates to
wait indefinitely.
    poolConfig.setMaxWaitMillis(redisPoolMaxWaitMillis);
    //Set to true to enable connectivity test on creating connections. Default: false.
    poolConfig.setTestOnCreate(false);
    //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
    poolConfig.setTestOnBorrow(true);
    //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
    poolConfig.setTestOnReturn(false);
    //Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
    poolConfig.setTestWhileIdle(true);
    //Duration after which idle connections are evicted. If the idle duration is greater than this value
and the maximum number of idle connections is reached, idle connections are directly evicted.
    poolConfig.setSoftMinEvictableIdleTimeMillis(redisPoolSoftMinEvictableIdleTimeMillis);
    //Disable MinEvictableIdleTimeMillis().
    poolConfig.setMinEvictableIdleTimeMillis(-1);
    //Interval for checking and evicting idle connections. Default: 60s.
    poolConfig.setTimeBetweenEvictionRunsMillis(redisPoolBetweenEvictionRunsMillis);
    return poolConfig;
}
}
```

## Parameter Description

**Table 2-1** RedisStandaloneConfiguration parameters

Parameter	Default Value	Description
hostName	localhost	IP address/domain name for connecting to a DCS Redis instance
port	6379	Port number
database	0	Database number. Default: 0.
password	-	Redis instance password

**Table 2-2** RedisClusterConfiguration parameters

Parameter	Description
clusterNodes	Cluster node connection information, including the node IP address and port number
maxRedirects	Maximum redirecting times
password	Password

**Table 2-3** JedisPoolConfig parameters

Parameter	Default Value	Description
minIdle	-	Minimum connections in the connection pool
maxIdle	-	Maximum idle connections in the connection pool
maxTotal	-	Maximum total connections in the connection pool
blockWhenExhausted	true	Indicates whether to wait after the connection pool is exhausted. <b>true</b> : Wait. <b>false</b> : Do not wait. To validate <b>maxWaitMillis</b> , this parameter must be set to <b>true</b> .
maxWaitMillis	-1	Maximum amount of time (in milliseconds) to wait for connection after the connection pool is exhausted. The default value -1 indicates to wait indefinitely.

Parameter	Default Value	Description
testOnCreate	false	Indicates whether to enable connectivity test on creating connections. <b>false</b> : Disable. <b>true</b> : Enable.
testOnBorrow	false	Indicates whether to enable connectivity test on obtaining connections. <b>false</b> : Disable. <b>true</b> : Enable. For heavy-traffic services, set this parameter to <b>false</b> to reduce overhead.
testOnReturn	false	Indicates whether to enable connectivity test on returning connections. <b>false</b> : Disable. <b>true</b> : Enable. For heavy-traffic services, set this parameter to <b>false</b> to reduce overhead.
testWhileIdle	false	Indicates whether to check for idle connections. If this parameter is set to <b>false</b> , idle connections are not evicted. Recommended value: <b>true</b> .
softMinEvictableIdleTimeMillis	1800000	Duration (in milliseconds) after which idle connections are evicted. If the idle duration is greater than this value and the maximum number of idle connections is reached, idle connections are directly evicted.
minEvictableIdleTimeMillis	60000	Minimum amount of time (in milliseconds) a connection may remain idle in the pool before it is eligible for eviction. The recommended value is <b>-1</b> , indicating that <b>softMinEvictableIdleTimeMillis</b> is used instead.
timeBetweenEvictionRunsMillis	60000	Interval (in milliseconds) for checking and evicting idle connections.

**Table 2-4** JedisClientConfiguration parameters

Parameter	Default Value	Description
connectTimeout	2000	Connection timeout interval, in milliseconds.
readTimeout	2000	Timeout interval for waiting for a response, in milliseconds.
poolConfig	-	Pool configurations. For details, see <a href="#">JedisPoolConfig</a> .

## Suggestion for Configuring DCS Instances

- Connection pool configuration

### NOTE

The following calculation is applicable only to common service scenarios. You can customize it based on your service requirements.

There is no standard connection pool size. You can configure one based on your service traffic. The following formulas are for reference:

- Minimum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command)
- Maximum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command) x 150%

For example, if the QPS of a service application is about 10,000, each request needs to access Redis 10 times (that is, 100,000 accesses to Redis every second), and the service application has 10 hosts, the calculation is as follows:

QPS of a single node accessing Redis = 100,000/10 = 10,000

Average time spent on a single command = 20 ms (Redis takes 5 ms to 10 ms to process a single command under normal conditions. If network jitter occurs, it takes 15 ms to 20 ms.)

Minimum number of connections = 10,000/(1000 ms/20 ms) = 200

Maximum number of connections = 10,000/(1000 ms/20 ms) x 150% = 300

### 2.2.3.1.2 Connecting to Redis on Lettuce (Java)

This section describes how to access a Redis instance on Lettuce. For more information about how to use other Redis clients, visit [the Redis official website](#).

Spring Data Redis is already integrated with [Jedis](#) and [Lettuce](#) for Spring Boot projects. In addition, Spring Boot 1.x is integrated with Jedis, and Spring Boot 2.x with Lettuce. Therefore, you do not need to import Lettuce in Spring Boot 2.x and later projects.

### NOTE

Springboot 2.3.12.RELEASE or later is required. Lettuce [6.3.0.RELEASE](#) or later is required.

## Prerequisites

- A Redis instance is created, and is in the **Running** state.
- View the IP address/domain name and port number of the DCS Redis instance to be accessed. For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

## Pom Configuration

```
<!-- Enable Spring Data Redis, Lettuce-supported SDK is integrated by default -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
  <groupId>io.lettuce</groupId>
  <artifactId>lettuce-core</artifactId>
```

```
<version>${lettuce.version}</version>  
</dependency>
```

## application.properties Configuration

- Single-node, master/standby, and Proxy Cluster

```
#Redis host  
spring.redis.host=<host>  
#Redis port  
spring.redis.port=<port>  
#Redis database number  
spring.redis.database=0  
#Redis password  
spring.redis.password=<password>  
#Redis read/write timeout  
spring.redis.timeout=2000
```

- Redis Cluster

```
#Redis Cluster node information  
spring.redis.cluster.nodes=<ip:port>,<ip:port>,<ip:port>  
#Redis Cluster max redirecting times  
spring.redis.cluster.max-redirects=3  
#Redis Cluster node password  
spring.redis.password=<password>  
#Redis Cluster timeout  
spring.redis.timeout=2000  
#Enable adaptive topology refresh  
spring.redis.lettuce.cluster.refresh.adaptive=true  
#Enable topology refresh every 10 seconds  
spring.redis.lettuce.cluster.refresh.period=10S
```

## Bean Configuration

- Single-node, master/standby, and Proxy Cluster

```
import java.time.Duration;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.connection.RedisConnectionFactory;  
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;  
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;  
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;  
  
import io.lettuce.core.ClientOptions;  
import io.lettuce.core.SocketOptions;  
  
/**  
 * Lettuce non-pooling configuration (use either this or the application.properties configuration)  
 */  
@Configuration  
public class RedisConfiguration {  
  
    @Value("${redis.host}")  
    private String redisHost;  
  
    @Value("${redis.port:6379}")  
    private Integer redisPort = 6379;  
  
    @Value("${redis.database:0}")  
    private Integer redisDatabase = 0;  
  
    @Value("${redis.password}")  
    private String redisPassword;  
  
    @Value("${redis.connect.timeout:2000}")  
    private Integer redisConnectTimeout = 2000;
```

```
@Value("${redis.read.timeout:2000}")
private Integer redisReadTimeout = 2000;

@Bean
public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

    RedisStandaloneConfiguration standaloneConfiguration = new RedisStandaloneConfiguration();
    standaloneConfiguration.setHostName(redisHost);
    standaloneConfiguration.setPort(redisPort);
    standaloneConfiguration.setDatabase(redisDatabase);
    standaloneConfiguration.setPassword(redisPassword);

    LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(standaloneConfiguration, clientConfiguration);
    connectionFactory.setDatabase(redisDatabase);
    return connectionFactory;
}

@Bean
public LettuceClientConfiguration clientConfiguration() {

    SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

    ClientOptions clientOptions = ClientOptions.builder()
        .autoReconnect(true)
        .pingBeforeActivateConnection(true)
        .cancelCommandsOnReconnectFailure(false)
        .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
        .socketOptions(socketOptions)
        .build();

    LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
        .commandTimeout(Duration.ofMillis(redisReadTimeout))
        .readFrom(ReadFrom.MASTER)
        .clientOptions(clientOptions)
        .build();

    return clientConfiguration;
}
}
```

- Pooling configuration for single-node, master/standby, and Proxy Cluster instances

#### Enable the pooling component

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
  <version>2.11.1</version>
</dependency>
```

#### Code

```
import java.time.Duration;

import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfiguration;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
```

```
/**
 * Lettuce pooling configuration
 */
@Configuration
public class RedisPoolConfiguration {
    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:2000}")
    private Integer redisPoolMaxWaitMillis = 2000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

        RedisStandaloneConfiguration standaloneConfiguration = new RedisStandaloneConfiguration();
        standaloneConfiguration.setHostName(redisHost);
        standaloneConfiguration.setPort(redisPort);
        standaloneConfiguration.setDatabase(redisDatabase);
        standaloneConfiguration.setPassword(redisPassword);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(standaloneConfiguration, clientConfiguration);
        connectionFactory.setDatabase(redisDatabase);
        //Disable sharing native connection before enabling pooling
        connectionFactory.setShareNativeConnection(false);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

        ClientOptions clientOptions = ClientOptions.builder()
            .autoReconnect(true)
            .pingBeforeActivateConnection(true)
            .cancelCommandsOnReconnectFailure(false)
            .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
            .socketOptions(socketOptions)
            .build();
    }
}
```

```
    LettucePoolingClientConfiguration clientConfiguration =
LettucePoolingClientConfiguration.builder()
    .poolConfig(poolConfig())
    .commandTimeout(Duration.ofMillis(redisReadTimeout))
    .clientOptions(clientOptions)
    .readFrom(ReadFrom.MASTER)
    .build();
    return poolingClientConfiguration;
}

private GenericObjectPoolConfig redisPoolConfig() {
    GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
    //Minimum idle connections in the pool
    poolConfig.setMinIdle(redisPoolMinSize);
    //Maximum idle connections in the pool
    poolConfig.setMaxIdle(redisPoolMaxSize);
    //Maximum total connections in the pool
    poolConfig.setMaxTotal(redisPoolMaxSize);
    //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
    poolConfig.setBlockWhenExhausted(true);
    //Max allowed time to wait for connection after pool is exhausted. The default value -1 indicates
to wait indefinitely.
    poolConfig.setMaxWait(Duration.ofMillis(redisPoolMaxWaitMillis));
    //Set to true to enable connectivity test on creating connections. Default: false.
    poolConfig.setTestOnCreate(false);
    //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
    poolConfig.setTestOnBorrow(true);
    //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
    poolConfig.setTestOnReturn(false);
    //Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
    poolConfig.setTestWhileIdle(true);
    //Idle duration after which a connection is evicted. If the actual duration is greater than this
value and the maximum number of idle connections is reached, idle connections are directly evicted.
    poolConfig.setSoftMinEvictableIdleTime(Duration.ofMillis(redisPoolSoftMinEvictableIdleTimeMillis));
    //Disable eviction policy MinEvictableIdleTimeMillis().
    poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
    //Interval for checking and evicting idle connections. Default: 60s.
    poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictionRunsMillis));
    return poolConfig;
}
}
```

- **Configuration for Redis Cluster instances**

```
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
import io.lettuce.core.cluster.ClusterClientOptions;
import io.lettuce.core.cluster.ClusterTopologyRefreshOptions;

/**
 * Lettuce Cluster non-pooling configuration (use either this or the application.properties configuration)
 */
```



```
@Configuration
public class RedisConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.cluster.maxDirects:3}")
    private Integer redisClusterMaxDirects;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.cluster.topology.refresh.period.millis:10000}")
    private Integer redisClusterTopologyRefreshPeriodMillis = 10000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0], Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(redisClusterMaxDirects);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(clusterConfiguration, clientConfiguration);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

        ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
            .enableAllAdaptiveRefreshTriggers()
            .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
            .build();

        ClusterClientOptions clientOptions = ClusterClientOptions.builder()
            .autoReconnect(true)
            .pingBeforeActivateConnection(true)
            .cancelCommandsOnReconnectFailure(false)
            .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
            .socketOptions(socketOptions)
            .topologyRefreshOptions(topologyRefreshOptions)
            .build();

        LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
            .commandTimeout(Duration.ofMillis(redisReadTimeout))
            .readFrom(ReadFrom.MASTER)
            .clientOptions(clientOptions)
            .build();
    }
}
```

```
    return clientConfiguration;
  }
}
```

- Pooling configuration for Redis Cluster instances

#### Enable the pooling component

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
  <version>2.11.1</version>
</dependency>
```

#### Code

```
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfiguration;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
import io.lettuce.core.cluster.ClusterClientOptions;
import io.lettuce.core.cluster.ClusterTopologyRefreshOptions;

/**
 * Lettuce pooling configuration
 */
@Configuration
public class RedisPoolConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.cluster.maxDirects:3}")
    private Integer redisClusterMaxDirects;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.cluster.topology.refresh.period.millis:10000}")
    private Integer redisClusterTopologyRefreshPeriodMillis = 10000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:2000}")
    private Integer redisPoolMaxWaitMillis = 2000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;
```

```
@Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

@Bean
public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

    RedisClusterConfiguration clusterConfiguration = new RedisClusterConfiguration();

    List<RedisNode> clusterNodes = new ArrayList<>();
    for (String clusterNodeStr : redisClusterNodes.split(",")) {
        String[] nodeInfo = clusterNodeStr.split(":");
        clusterNodes.add(new RedisNode(nodeInfo[0], Integer.valueOf(nodeInfo[1])));
    }
    clusterConfiguration.setClusterNodes(clusterNodes);

    clusterConfiguration.setPassword(redisPassword);
    clusterConfiguration.setMaxRedirects(redisClusterMaxDirects);

    LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(clusterConfiguration, clientConfiguration);
    //Disable native connection sharing before validating connection pool
    connectionFactory.setShareNativeConnection(false);
    return connectionFactory;
}

@Bean
public LettuceClientConfiguration clientConfiguration() {

    SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

    ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
        .enableAllAdaptiveRefreshTriggers()
        .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
        .build();

    ClusterClientOptions clientOptions = ClusterClientOptions.builder()
        .autoReconnect(true)
        .pingBeforeActivateConnection(true)
        .cancelCommandsOnReconnectFailure(false)
        .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
        .socketOptions(socketOptions)
        .topologyRefreshOptions(topologyRefreshOptions)
        .build();

    LettucePoolingClientConfiguration clientConfiguration =
LettucePoolingClientConfiguration.builder()
        .poolConfig(poolConfig())
        .commandTimeout(Duration.ofMillis(redisReadTimeout))
        .clientOptions(clientOptions)
        .readFrom(ReadFrom.MASTER)
        .build();
    return clientConfiguration;
}

private GenericObjectPoolConfig poolConfig() {
    GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
    //Minimum connections in the pool
    poolConfig.setMinIdle(redisPoolMinSize);
    //Maximum idle connections in the pool
    poolConfig.setMaxIdle(redisPoolMaxSize);
    //Maximum total connections in the pool
    poolConfig.setMaxTotal(redisPoolMaxSize);
    //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
    poolConfig.setBlockWhenExhausted(true);
    //Max allowed time to wait for connection after pool is exhausted. The default value -1 indicates
```

```

to wait indefinitely.
    poolConfig.setMaxWait(Duration.ofMillis(redisPoolMaxWaitMillis));
    //Set to true to enable connectivity test on creating connections. Default: false.
    poolConfig.setTestOnCreate(false);
    //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
    poolConfig.setTestOnBorrow(true);
    //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
    poolConfig.setTestOnReturn(false);
    //Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
    poolConfig.setTestWhileIdle(true);
    //Disable connection closure when the minimum idle time is reached.
    poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
    //Idle duration before a connection being evicted. If the actual duration is greater than this value
and the maximum number of idle connections is reached, idle connections are directly evicted.
MinEvictableIdleTimeMillis (default eviction policy) is no longer used.

poolConfig.setSoftMinEvictableIdleTime(Duration.ofMillis(redisPoolSoftMinEvictableIdleTimeMillis));
//Interval for checking and evicting idle connections. Default: 60s.
poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictionRunsMillis));

return poolConfig;
}
}

```

## Parameter Description

**Table 2-5** LettuceConnectionFactory parameters

Parameter	Type	Default Value	Description
configuration	RedisConfiguration	-	Redis connection configuration. Two subclasses: <ul style="list-style-type: none"> <li>RedisStandaloneConfiguration</li> <li>RedisClusterConfiguration</li> </ul>
clientConfiguration	LettuceClientConfiguration	-	Client configuration parameter. Common subclass: LettucePoolingClientConfiguration
shareNativeConnection	boolean	true	Indicates whether to share native connections. Set to <b>true</b> to share. Set to <b>false</b> to enable connection pooling.

**Table 2-6** RedisStandaloneConfiguration parameters

Parameter	Default Value	Description
hostName	localhost	IP address/domain name for connecting to a DCS Redis instance

Parameter	Default Value	Description
port	6379	Port number
database	0	Database subscript
password	-	Password

**Table 2-7** RedisClusterConfiguration parameters

Parameter	Description
clusterNodes	Cluster node connection information, including the node IP address and port number
maxRedirects	Maximum redirecting times. Recommended value: <b>3</b> .
password	Password

**Table 2-8** LettuceClientConfiguration parameters

Parameter	Type	Default Value	Description
timeout	Duration	60s	Command timeout: Recommended: <b>2s</b> .
clientOptions	ClientOptions	-	Configuration options.
readFrom	readFrom	MASTER	Read mode. Recommended: <b>MASTER</b> . Other values may cause access failures in failover scenarios.

**Table 2-9** LettucePoolingClientConfiguration parameters

Parameter	Type	Default Value	Description
timeout	Duration	60s	Command timeout: Recommended: <b>2s</b> .
clientOptions	ClientOptions	-	Configuration options.
poolConfig	GenericObjectPoolConfig	-	Connection pool configuration.

Parameter	Type	Default Value	Description
readFrom	readFrom	MASTER	Read mode. Recommended: <b>MASTER</b> . Other values may cause access failures in failover scenarios.

**Table 2-10** ClientOptions parameters

Parameter	Type	Default Value	Description
autoReconnect	boolean	true	Indicates whether to automatically reconnect after disconnection. Recommended: <b>true</b> .
pingBeforeActivateConnection	boolean	true	Indicates whether to test connectivity on established connections. Recommended: <b>true</b> .
cancelCommandsOnReconnectFailure	boolean	true	Indicates whether to cancel commands after a failed reconnection attempt. Recommended: <b>false</b> .
disconnectedBehavior	DisconnectedBehavior	DisconnectedBehavior.DEFAULT	Indicates what to do when a connection drops. Recommended: <b>ACCEPT_COMMANDS</b> . <ul style="list-style-type: none"> <li>• <b>DEFAULT</b>: When <b>autoReconnect</b> is set <b>true</b>, commands are allowed to wait in queue. When <b>autoReconnect</b> is set to <b>false</b>, commands are not allowed to wait in queue.</li> <li>• <b>ACCEPT_COMMANDS</b>: Allow commands to wait in queue.</li> <li>• <b>REJECT_COMMANDS</b>: Do not allow commands to wait in queue.</li> </ul>
socketOptions	SocketOptions	-	Socket configuration.

**Table 2-11** SocketOptions parameters

Parameter	Default Value	Description
connectTimeout	10s	Connection timeout. Recommended: <b>2s</b> .

**Table 2-12** GenericObjectPoolConfig parameters

Parameter	Default Value	Description
minIdle	-	Minimum connections in the pool.
maxIdle	-	Maximum idle connections in the connection pool.
maxTotal	-	Maximum total connections in the connection pool.
blockWhenExhausted	true	Indicates whether to wait after the connection pool is exhausted. <b>true</b> : Wait. <b>false</b> : Do not wait. To validate <b>maxWaitMillis</b> , this parameter must be set to <b>true</b> .
maxWaitMillis	-1	Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. The default value <b>-1</b> indicates to wait indefinitely.
testOnCreate	false	Set to true to enable connectivity test on creating connections. Default: <b>false</b> .
testOnBorrow	false	Set to true to enable connectivity test on borrowing connections. Default: <b>false</b> . Set to false for heavy-traffic services to reduce overhead.
testOnReturn	false	Set to <b>true</b> to enable connectivity test on returning connections. Default: <b>false</b> . Set to <b>false</b> for heavy-traffic services to reduce overhead.
testWhileIdle	false	Indicates whether to check for idle connections. If this parameter is set to <b>false</b> , idle connections are not evicted. Recommended value: <b>true</b> .
softMinEvictableIdleTimeMillis	1800000	Duration after which idle connections are evicted. If the idle duration is greater than this value and the maximum number of idle connections is reached, idle connections are directly evicted.

Parameter	Default Value	Description
minEvictableIdleTimeMillis	60000	An eviction policy, set to <b>-1</b> (suggested) to disable it. Use <b>softminEvictableIdleTimeMillis</b> instead.
timeBetweenEvictionRunsMillis	60000	Eviction interval, in milliseconds.

## Suggestion for Configuring DCS Instances

- Pooling connection

Different from Jedis's BIO, the bottom layer of Lettuce communicates with Redis Server based on Netty's NIO. Combining persistent connections and queues, Lettuce sends and receives multiple requests and responses spontaneously with sequential sending and receiving features of TCP. A single connection supports 3000 to 5000 QPS, but you are not advised to allow more than 3000 QPS in production systems. Pooling is not supported by Lettuce, and is disabled by default in Spring Boot. To enable pooling, validate the commons-pool2 dependency and disable native connection sharing.

By default, each Lettuce connection needs two thread pools, I/O thread pool and computation thread pool, to support I/O event reading and asynchronous event processing. If you configure connection pooling, each connection creates two thread pools, consuming high memory resources. **Lettuce is strong at processing single connections based on its bottom-layer implementation, so you are not advised to use Lettuce with pooling.**
- Topology refresh

When connecting to a Redis Cluster instance, Lettuce randomly sends **cluster nodes** to the node list during initialization to obtain the distribution of cluster slots. Cluster topology structure changes when the cluster capacity is increased or decreased or a master/standby switchover occurs. Lettuce does not detect such changes by default. You can enable detection with the following configurations:

  - application.properties configuration**

```
#Enable adaptive topology refresh.
spring.redis.lettuce.cluster.refresh.adaptive=true
#Enable topology refresh every 10 seconds.
spring.redis.lettuce.cluster.refresh.period=10S
```
  - API configuration**

```
ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
    .enableAllAdaptiveRefreshTriggers()
    .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
    .build();

ClusterClientOptions clientOptions = ClusterClientOptions.builder()
    ...
    ...
    .topologyRefreshOptions(topologyRefreshOptions)
    .build();
```
- Blast radius

The bottom layer of Lettuce uses a combination of single persistent connection and request queue. Once network jitter or intermittent



disconnection occurs or connection times out, all requests are affected. Especially when connection times out, an attempt is made to resend TCP packets until timeout and connection drops. Requests do not recover until connections are reestablished. Requests accumulate during resending attempts. If upper-layer services time out in batches, or the resending timeout is too long in some OSs' kernels, the service system remains unavailable for a long time. **Therefore, you are advised to use Jedis instead of Lettuce.**

### 2.2.3.1.3 Connecting to Redis on Redisson (Java)

This section describes how to access a Redis instance on Redisson. For more information about how to use other Redis clients, visit [the Redis official website](#).

For Spring Boot projects, Spring Data Redis is already integrated with [Jedis](#) and [Lettuce](#), but does not support Redisson. [Redisson](#) provides the redisson-spring-boot-starter component (<https://mvnrepository.com/artifact/org.redisson/redisson>) that can be used with Spring Boot.

Spring Boot 1.x is integrated with Jedis, and Spring Boot 2.x is integrated with Lettuce.

#### NOTE

- If a password was set during DCS Redis instance creation, configure the password for connecting to Redis using Redisson. Do not hard code the plaintext password.
- To connect to a single-node or Proxy Cluster instance, use the `useSingleServer` method of the `SingleServerConfig` object of Redisson. To connect to a master/standby instance, use the `useMasterSlaveServers` method of the `MasterSlaveServersConfig` object of Redisson. To connect to a Redis Cluster instance, use the `useClusterServers` method of the `ClusterServersConfig` object.
- Springboot 2.3.12.RELEASE or later is required. Redisson [3.37.0](#) or later is required.

## Prerequisites

- A Redis instance is created, and is in the **Running** state.
- View the IP address/domain name and port number of the DCS Redis instance to be accessed. For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

## Pom Configuration

```
<!-- spring-data-redis -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
  <exclusions>
    <!--Lettuce is integrated in Spring Boot 2.x by default. This dependency needs to be deleted. -->
    <exclusion>
      <artifactId>lettuce-core</artifactId>
      <groupId>io.lettuce</groupId>
    </exclusion>
  </exclusions>
</dependency>
<!--Redisson's adaptation package for Spring Boot-->
<dependency>
  <groupId>org.redisson</groupId>
  <artifactId>redisson-spring-boot-starter</artifactId>
  <version>${redisson.version}</version>
</dependency>
```

## Bean Configuration

Spring Boot does not provide Redisson adaptation, and the **application.properties** configuration file does not have the corresponding configuration item. Therefore, you can only use Bean configuration.

- Single-node and Proxy Cluster

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.SingleServerConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SingleConfig {

    @Value("${redis.address}")
    private String redisAddress;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.connection.pool.min.size:50}")
    private Integer redisConnectionPoolMinSize;

    @Value("${redis.connection.pool.max.size:200}")
    private Integer redisConnectionPoolMaxSize;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Bean
    public RedissonClient redissonClient(){
        Config redissonConfig = new Config();

        SingleServerConfig serverConfig = redissonConfig.useSingleServer();
        serverConfig.setAddress(redisAddress);
        serverConfig.setConnectionMinimumIdleSize(redisConnectionPoolMinSize);
        serverConfig.setConnectionPoolSize(redisConnectionPoolMaxSize);

        serverConfig.setDatabase(redisDatabase);
        serverConfig.setPassword(redisPassword);
        serverConfig.setConnectTimeout(redisConnectTimeout);
        serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
        serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
        serverConfig.setTimeout(timeout);
        serverConfig.setRetryAttempts(redisRetryAttempts);
```

```
serverConfig.setRetryInterval(redisRetryInterval);

redissonConfig.setCodec(new JsonJacksonCodec());
return Redisson.create(redissonConfig);
}
}
```

- **Master/Standby**

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.MasterSlaveServersConfig;
import org.redisson.config.ReadMode;
import org.redisson.config.SubscriptionMode;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashSet;

@Configuration
public class MasterStandbyConfig {
    @Value("${redis.master.address}")
    private String redisMasterAddress;

    @Value("${redis.slave.address}")
    private String redisSlaveAddress;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.master.connection.pool.min.size:50}")
    private Integer redisMasterConnectionPoolMinSize = 50;

    @Value("${redis.master.connection.pool.max.size:200}")
    private Integer redisMasterConnectionPoolMaxSize = 200;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Bean
    public RedissonClient redissonClient() {
        Config redissonConfig = new Config();

        MasterSlaveServersConfig serverConfig = redissonConfig.useMasterSlaveServers();
        serverConfig.setMasterAddress(redisMasterAddress);
        HashSet<String> slaveSet = new HashSet<>();
        slaveSet.add(redisSlaveAddress);
        serverConfig.setSlaveAddresses(slaveSet);

        serverConfig.setDatabase(redisDatabase);
    }
}
```

```
serverConfig.setPassword(redisPassword);

serverConfig.setMasterConnectionMinimumIdleSize(redisMasterConnectionPoolMinSize);
serverConfig.setMasterConnectionPoolSize(redisMasterConnectionPoolMaxSize);

serverConfig.setReadMode(ReadMode.MASTER);
serverConfig.setSubscriptionMode(SubscriptionMode.MASTER);

serverConfig.setConnectTimeout(redisConnectTimeout);
serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
serverConfig.setTimeout(timeout);
serverConfig.setRetryAttempts(redisRetryAttempts);
serverConfig.setRetryInterval(redisRetryInterval);

redissonConfig.setCodec(new JsonJacksonCodec());
return Redisson.create(redissonConfig);
}
}
```

- **Redis Cluster**

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.ClusterServersConfig;
import org.redisson.config.Config;
import org.redisson.config.ReadMode;
import org.redisson.config.SubscriptionMode;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.List;

@Configuration
public class ClusterConfig {

    @Value("${redis.cluster.address}")
    private List<String> redisClusterAddress;

    @Value("${redis.cluster.scan.interval:5000}")
    private Integer redisClusterScanInterval = 5000;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Value("${redis.master.connection.pool.min.size:50}")
    private Integer redisMasterConnectionPoolMinSize = 50;

    @Value("${redis.master.connection.pool.max.size:200}")
    private Integer redisMasterConnectionPoolMaxSize = 200;
}
```

```

@Bean
public RedissonClient redissonClient() {
    Config redissonConfig = new Config();

    ClusterServersConfig serverConfig = redissonConfig.useClusterServers();
    serverConfig.setNodeAddresses(redisClusterAddress);
    serverConfig.setScanInterval(redisClusterScanInterval);

    serverConfig.setPassword(redisPassword);

    serverConfig.setMasterConnectionMinimumIdleSize(redisMasterConnectionPoolMinSize);
    serverConfig.setMasterConnectionPoolSize(redisMasterConnectionPoolMaxSize);

    serverConfig.setReadMode(ReadMode.MASTER);
    serverConfig.setSubscriptionMode(SubscriptionMode.MASTER);

    serverConfig.setConnectTimeout(redisConnectTimeout);
    serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
    serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
    serverConfig.setTimeout(timeout);
    serverConfig.setRetryAttempts(redisRetryAttempts);
    serverConfig.setRetryInterval(redisRetryInterval);

    redissonConfig.setCodec(new JsonJacksonCodec());
    return Redisson.create(redissonConfig);
}
    
```

## Parameter Description

**Table 2-13** Config parameters

Parameter	Default Value	Description
codec	org.redisson.codec.JsonJacksonCodec	Encoding format, including JSON, Avro, Smile, CBOR, and MsgPack.
threads	Number of CPU cores x 2	Thread pool used for executing RTopic Listener, RRemoteService, and RExecutorService.
executor	null	The function is the same as <b>threads</b> . If this parameter is not set, a thread pool is initialized based on <b>threads</b> .
nettyThreads	Number of CPU cores x 2	Thread pool used by the TCP channel that connects to the redis-server. All channels share this connection pool and are mapped to Netty's <b>Bootstrap.group(...)</b> .
eventLoopGroup	null	The function is the same as <b>nettyThreads</b> . If this parameter is not set, an EventLoopGroup is initialized based on the <b>nettyThreads</b> parameter for the bottom-layer TCP channel to use.
transportMode	TransportMode.NIO	Transmission mode. The options are <b>NIO</b> , <b>EPOLL</b> (additional package required), and <b>KQUEUE</b> (additional package required).

Parameter	Default Value	Description
lockWatchdogTimeout	30000	Timeout interval (in milliseconds) of the lock-monitoring watchdog. In the distributed lock scenario, if the <b>leaseTimeout</b> parameter is not specified, the default value of this parameter is used.
keepPubSubOrder	true	Indicates whether to receive messages in the publish sequence. <b>If messages can be processed concurrently, you are advised to set this parameter to false.</b>

**Table 2-14** SingleServerConfig parameters (single-node, or Proxy Cluster)

Parameter	Default Value	Description
address	-	Node connection information, in redis:// <i>ip:port</i> format.
database	0	ID of the database to be used.
connectionMinimumIdleSize	32	Minimum number of connections to the master node of each shard.
connectionPoolSize	64	Maximum number of connections to the master node of each shard.
subscriptionConnectionMinimumIdleSize	1	Minimum number of connections to the target node for pub/sub.
subscriptionConnectionPoolSize	50	Maximum number of connections to the target node for pub/sub.
subscriptionPerConnection	5	Maximum number of subscriptions on each subscription connection.
connectionTimeout	10000	Connection timeout interval, in milliseconds.
idleConnectionTimeout	10000	Maximum time (in milliseconds) for reclaiming idle connections.
pingConnectionInterval	30000	Heartbeat for detecting available connections, in milliseconds. <b>Recommended: 3000 ms.</b>
timeout	3000	Timeout interval for waiting for a response, in milliseconds.
retryAttempts	3	Maximum number of retries upon send failures.

Parameter	Default Value	Description
retryInterval	1500	Retry interval, in milliseconds. <b>Recommended: 200 ms.</b>
clientName	null	Client name.

**Table 2-15** MasterSlaveServersConfig parameters (master/standby)

Parameter	Default Value	Description
masterAddress	-	Master node connection information, in <code>redis://ip:port</code> format.
slaveAddresses	-	Standby node connection information, in <code>Set&lt;redis://ip:port&gt;</code> format.
readMode	SLAVE	Read mode. By default, read traffic is distributed to replica nodes. The value can be <b>MASTER</b> (recommended), <b>SLAVE</b> , or <b>MASTER_SLAVE</b> . Other values may cause access failures in failover scenarios.
loadBalancer	RoundRobinLoad Balancer	Load balancing algorithm. This parameter is valid only when <b>readMode</b> is set to <b>SLAVE</b> or <b>MASTER_SLAVE</b> . Read traffic is distributed evenly.
masterConnectionMinimumIdleSize	32	Minimum number of connections to the master node of each shard.
masterConnectionPoolSize	64	Maximum number of connections to the master node of each shard.
slaveConnectionMinimumIdleSize	32	Minimum number of connections to each replica node of each shard. If <b>readMode</b> is set to <b>MASTER</b> , the value of this parameter is invalid.
slaveConnectionPoolSize	64	Maximum number of connections to each replica node of each shard. If <b>readMode</b> is set to <b>MASTER</b> , the value of this parameter is invalid.
subscriptionMode	SLAVE	Subscription mode. By default, only replica nodes handle subscription. The value can be <b>SLAVE</b> or <b>MASTER</b> (recommended).
subscriptionConnectionMinimumIdleSize	1	Minimum number of connections to the target node for pub/sub.

Parameter	Default Value	Description
subscriptionConnectionPoolSize	50	Maximum number of connections to the target node for pub/sub.
subscriptionPerConnection	5	Maximum number of subscriptions on each subscription connection.
connectionTimeout	10000	Connection timeout interval, in milliseconds.
idleConnectionTimeout	10000	Maximum time (in milliseconds) for reclaiming idle connections.
pingConnectionInterval	30000	Heartbeat for detecting available connections, in milliseconds. <b>Recommended: 3000 ms.</b>
timeout	3000	Timeout interval for waiting for a response, in milliseconds.
retryAttempts	3	Maximum number of retries upon send failures.
retryInterval	1500	Retry interval, in milliseconds. <b>Recommended: 200 ms.</b>
clientName	null	Client name.

**Table 2-16** ClusterServersConfig parameters (Redis Cluster)

Parameter	Default Value	Description
nodeAddress	-	Connection addresses of cluster nodes. Each address uses the <code>redis://ip:port</code> format. Use commas (,) to separate connection addresses of different nodes.
password	null	Password for logging in to the cluster.
scanInterval	1000	Interval for periodically checking the cluster node status, in milliseconds.
readMode	SLAVE	Read mode. By default, read traffic is distributed to replica nodes. The value can be <b>MASTER</b> (recommended), <b>SLAVE</b> , or <b>MASTER_SLAVE</b> . Other values may cause access failures in failover scenarios.
loadBalancer	RoundRobinLoadBalancer	Load balancing algorithm. This parameter is valid only when <b>readMode</b> is set to <b>SLAVE</b> or <b>MASTER_SLAVE</b> . Read traffic is distributed evenly.



Parameter	Default Value	Description
masterConnectionMinimumIdleSize	32	Minimum number of connections to the master node of each shard.
masterConnectionPoolSize	64	Maximum number of connections to the master node of each shard.
slaveConnectionMinimumIdleSize	32	Minimum number of connections to each replica node of each shard. If <b>readMode</b> is set to <b>MASTER</b> , the value of this parameter is invalid.
slaveConnectionPoolSize	64	Maximum number of connections to each replica node of each shard. If <b>readMode</b> is set to <b>MASTER</b> , the value of this parameter is invalid.
subscriptionMode	SLAVE	Subscription mode. By default, only replica nodes handle subscription. The value can be <b>SLAVE</b> or <b>MASTER</b> (recommended).
subscriptionConnectionMinimumIdleSize	1	Minimum number of connections to the target node for pub/sub.
subscriptionConnectionPoolSize	50	Maximum number of connections to the target node for pub/sub.
subscriptionPerConnection	5	Maximum number of subscriptions on each subscription connection.
connectionTimeout	10000	Connection timeout interval, in milliseconds.
idleConnectionTimeout	10000	Maximum time (in milliseconds) for reclaiming idle connections.
pingConnectionInterval	30000	Heartbeat for detecting available connections, in milliseconds. <b>Recommended: 3000.</b>
timeout	3000	Timeout interval for waiting for a response, in milliseconds.
retryAttempts	3	Maximum number of retries upon send failures.
retryInterval	1500	Retry interval, in milliseconds. <b>Recommended: 200.</b>
clientName	null	Client name.

## Suggestion for Configuring DCS Instances

- **readMode**  
**MASTER** is the recommended value, that is, the master node bears all read and write traffic. This is to avoid data inconsistency caused by master/replica synchronization latency. If the value is **SLAVE**, all read requests will trigger errors when replicas are faulty. If the value is **MASTER\_SLAVE**, some read requests will trigger errors. Read errors last for the period specified by **failedSlaveCheckInterval** (180s by default) until the faulty nodes are removed from the available node list.
- **subscriptionMode**  
Similar to **readMode**, **MASTER** is the recommended value.
- Connection pool configuration

### NOTE

The following calculation is applicable only to common service scenarios. You can customize it based on your service requirements.

There is no standard connection pool size. You can configure one based on your service traffic. The following formulas are for reference:

- Minimum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command)
- Maximum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command) x 150%

For example, if the QPS of a service application is about 10,000, each request needs to access Redis 10 times (that is, 100,000 accesses to Redis every second), and the service application has 10 hosts, the calculation is as follows:

QPS of a single node accessing Redis = 100,000/10 = 10,000

Average time spent on a single command = 20 ms (Redis takes 5 ms to 10 ms to process a single command under normal conditions. If network jitter occurs, it takes 15 ms to 20 ms.)

Minimum number of connections = 10,000/(1000 ms/20 ms) = 200

Maximum number of connections = 10,000/(1000 ms/20 ms) x 150% = 300

- Retry configuration  
Redisson supports retries. You can set the following parameters based on service requirements. Generally, configure three retries, and set the retry interval to about 200 ms.
  - **retryAttempts**: number of retry times
  - **retryInterval**: retry interval

### NOTE

In Redisson, some APIs are implemented through LUA, and the performance is low. You are advised to use Jedis instead of Redisson.

### 2.2.3.2 Connecting to Redis on redis-py (Python)

This section describes how to access a Redis instance on redis-py. For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

 **NOTE**

Use `redis-py` to connect to single-node, master/standby, and Proxy Cluster instances and `redis-py-cluster` to connect to Redis Cluster instances.

## Prerequisites

- A Redis instance is created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*
- If the ECS runs the Linux OS, ensure that the Python compilation environment has been installed on the ECS.

## Connecting to Redis on `redis-py`

**Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

**Step 2** Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance using a Python client.

**Step 3** Access the DCS Redis instance.

If the ECS OS does not provide Python, run the following **yum** command to install it:

```
yum install python
```

 **NOTE**

The Python version must be 3.6 or later. If the default Python version is earlier than 3.6, perform the following operations to change it:

1. Run the **rm -rf python** command to delete the Python symbolic link.
2. Run the **ln -s python.X.X.X python** command to create another Python link. In the command, *X.X.X* indicates the Python version number.

- **For single-node, master/standby, or Proxy Cluster types:**

- a. Install Python and `redis-py`.
  - i. If the system does not provide Python, run the **yum** command to install it.
  - ii. Run the following command to download and decompress the `redis-py` package:

```
wget https://github.com/andymccurdy/redis-py/archive/master.zip  
unzip master.zip
```
  - iii. Go to the directory where the decompressed `redis-py` package is saved, and install `redis-py`.

```
python setup.py install
```

After the installation, run the **python** command. `redis-py` have been successfully installed if the following command output is displayed:

**Figure 2-1** Running the python command

```
[root@ecs-... redis-py-master]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

- b. Use the redis-py client to connect to the instance. In the following steps, commands are executed in CLI mode. (Alternatively, write the commands into a Python script and then execute the script.)
  - i. Run the **python** command to enter the CLI mode. You have entered CLI mode if the following command output is displayed:

**Figure 2-2** Entering the CLI mode

```
[root@ecs-... redis-py-master]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

- ii. Run the following command to access the chosen DCS Redis instance:

```
r = redis.StrictRedis(host='XXX.XXX.XXX.XXX', port=6379, password='*****');
```

*XXX.XXX.XXX.XXX* indicates the IP address/domain name of the DCS instance and **6379** is an example port number of the instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change them as required. *\*\*\*\*\** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

You have successfully accessed the instance if the following command output is displayed. Enter commands to perform read and write operations on the database.

**Figure 2-3** Redis connected successfully

```
>>> r = redis.StrictRedis(host='..._9', port=6379, password='...');
>>> r.set("foo", "bar")
True
>>> print(r.get("foo"))
b'bar'
>>> _
```

- **For the Redis Cluster type:**
  - a. Install the redis-py-cluster client.
    - i. Download the released version.

```
wget https://github.com/Grokzen/redis-py-cluster/releases/download/2.1.3/redis-py-cluster-2.1.3.tar.gz
```
    - ii. Decompress the package.

```
tar -xvf redis-py-cluster-2.1.3.tar.gz
```
    - iii. Go to the directory where the decompressed redis-py-cluster package is saved, and install redis-py-cluster.

```
python setup.py install
```
  - b. Access the DCS Redis instance by using redis-py-cluster.

In the following steps, commands are executed in CLI mode. (Alternatively, write the commands into a Python script and then execute the script.)

- i. Run the **python** command to enter the CLI mode.
- ii. Run the following command to access the chosen DCS Redis instance. If the instance does not have a password, exclude **password='\*\*\*\*\*'** from the command.

```
>>> from rediscluster import RedisCluster

>>> startup_nodes = [{"host": "192.168.0.143", "port": "6379"}, {"host": "192.168.0.144", "port": "6379"}, {"host": "192.168.0.145", "port": "6379"}, {"host": "192.168.0.146", "port": "6379"}]

>>> rc = RedisCluster(startup_nodes=startup_nodes, decode_responses=True, password='*****')

>>> rc.set("foo", "bar")
True
>>> print(rc.get("foo"))
'bar'
```

----End

### 2.2.3.3 Connecting to Redis on go-redis (Go)

This section describes how to access a Redis instance on go-redis. For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

#### Prerequisites

- A Redis instance is created, and is in the **Running** state.
- View the IP address/domain name and port number of the DCS Redis instance to be accessed. For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*

#### Connecting to Redis on go-redis

**Step 1** Log in to the ECS.

A Windows ECS is used as an example.

**Step 2** Install Visual Studio Community 2017 on the ECS.

**Step 3** Start Visual Studio and create a project. The project name can be customized. In this example, the project name is set to **redisdemo**.

**Step 4** Import the dependency package of go-redis and enter **go get github.com/go-redis/redis** on the terminal.

**Step 5** Write the following code:

```
package main

import (
    "fmt"
    "github.com/go-redis/redis"
)
```

```
func main() {
    // Single-node
    rdb := redis.NewClient(&redis.Options{
        Addr:     "host:port",
        Password: "*****", // no password set
        DB:      0, // use default DB
    })

    val, err := rdb.Get("key").Result()
    if err != nil {
        if err == redis.Nil {
            fmt.Println("key does not exists")
            return
        }
        panic(err)
    }
    fmt.Println(val)

    //Cluster
    rdbCluster := redis.NewClusterClient(&redis.ClusterOptions{
        Addrs: []string{"host:port"},
        Password: "*****",
    })
    val1, err1 := rdbCluster.Get("key").Result()
    if err1 != nil {
        if err == redis.Nil {
            fmt.Println("key does not exists")
            return
        }
        panic(err)
    }
    fmt.Println(val1)
}
```

*host:port* are the IP address/domain name and port number of the DCS Redis instance. For details about how to obtain the IP address/domain name and port, see [Prerequisites](#). Change them as required. *\*\*\*\*\** indicates the password used to log in to the DCS Redis instance. This password is defined during DCS Redis instance creation.

**Step 6** Run the **go build -o test main.go** command to package the code into an executable file, for example, **test**.

---

**⚠ CAUTION**

To run the package in the Linux OS, set the following parameters before packaging:

**set GOARCH=amd64**

**set GOOS=linux**

---

**Step 7** Run the **./test** command to access the DCS instance.

----End

### 2.2.3.4 Connecting to Redis on hiredis (C++)

This section describes how to access a Redis instance on hiredis (C++). For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

 NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use C++ to connect to a Redis Cluster instance, see the [C++ Redis client description](#).

## Prerequisites

- A Redis instance is created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

## Connecting to Redis on hiredis

- Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

- Step 2** Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance in C++.

- Step 3** Install GCC, Make, and hiredis.

If the system does not provide a compiling environment, run the following **yum** command to install the environment:

```
yum install gcc make
```

- Step 4** Run the following command to download and decompress the hiredis package:

```
wget https://github.com/redis/hiredis/archive/master.zip  
unzip master.zip
```

- Step 5** Go to the directory where the decompressed hiredis package is saved, and compile and install hiredis.

```
make  
make install
```

- Step 6** Access the DCS instance by using hiredis.

The following describes connection and password authentication of hiredis. For more information on how to use hiredis, visit the Redis official website.

1. Edit the sample code for connecting to a DCS instance, and then save the code and exit.

```
vim connRedis.c
```

Example:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <hiredis.h>  
int main(int argc, char **argv) {  
    unsigned int j;  
    redisContext *conn;
```

```
redisReply *reply;
if (argc < 3) {
    printf("Usage: example {instance_ip_address} 6379 {password}\n");
    exit(0);
}
const char *hostname = argv[1];
const int port = atoi(argv[2]);
const char *password = argv[3];
struct timeval timeout = { 1, 500000 }; // 1.5 seconds
conn = redisConnectWithTimeout(hostname, port, timeout);
if (conn == NULL || conn->err) {
    if (conn) {
        printf("Connection error: %s\n", conn->errstr);
        redisFree(conn);
    } else {
        printf("Connection error: can't allocate redis context\n");
    }
    exit(1);
}
/* AUTH */
reply = redisCommand(conn, "AUTH %s", password);
printf("AUTH: %s\n", reply->str);
freeReplyObject(reply);

/* Set */
reply = redisCommand(conn, "SET %s %s", "welcome", "Hello, DCS for Redis!");
printf("SET: %s\n", reply->str);
freeReplyObject(reply);

/* Get */
reply = redisCommand(conn, "GET welcome");
printf("GET welcome: %s\n", reply->str);
freeReplyObject(reply);

/* Disconnects and frees the context */
redisFree(conn);
return 0;
}
```

2. Run the following command to compile the code:

```
gcc connRedis.c -o connRedis -I /usr/local/include/hiredis -lhiredis
```

If an error is reported, locate the directory where the **hiredis.h** file is saved and modify the compilation command.

After the compilation, an executable **connRedis** file is obtained.

3. Run the following command to access the chosen DCS Redis instance:

```
./connRedis {redis_instance_address} 6379 {password}
```

*{redis\_instance\_address}* indicates the IP address/domain name of DCS instance and **6379** is an example port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change them as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

You have successfully accessed the instance if the following command output is displayed:

```
AUTH: OK
SET: OK
GET welcome: Hello, DCS for Redis!
```



**NOTICE**

If an error is reported, indicating that the hiredis library files cannot be found, run the following commands to copy related files to the system directories and add dynamic links:

```
mkdir /usr/lib/hiredis
cp /usr/local/lib/libhiredis.so.0.13 /usr/lib/hiredis/
mkdir /usr/include/hiredis
cp /usr/local/include/hiredis/hiredis.h /usr/include/hiredis/
echo '/usr/local/lib' >> >> /etc/ld.so.conf
ldconfig
```

Replace the locations of the **so** and **.h** files with actual ones before running the commands.

----End

### 2.2.3.5 Connecting to Redis on StackExchange.Redis (C#)

This section describes how to access a Redis instance on StackExchange.Redis. For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

#### Prerequisites

- A Redis instance is created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

#### Connecting to Redis on StackExchange.Redis

**Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

**Step 2** Log in to the ECS.

A Windows ECS is used as an example.

**Step 3** Install Visual Studio Community 2017 on the ECS.

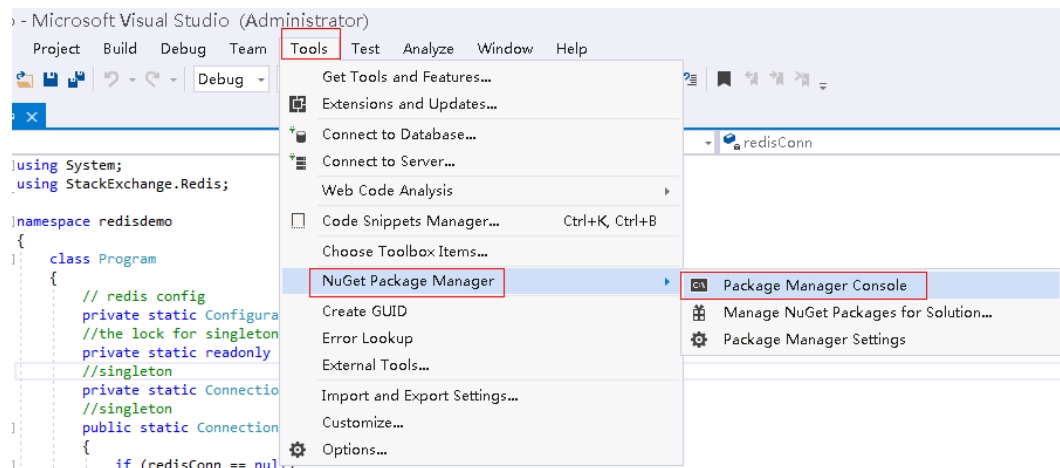
**Step 4** Start Visual Studio 2017 and create a project.

Set the project name to **redisdemo**.

**Step 5** Install StackExchange.Redis by using the NuGet package manager of Visual Studio.

Access the NuGet package manager console according to [Figure 2-4](#), and enter **Install-Package StackExchange.Redis -Version 2.2.79**. (The version number is optional).

**Figure 2-4** Accessing the NuGet package manager console



**Step 6** Write the following code, and use the String Set and Get methods to test the connection.

```
using System;
using StackExchange.Redis;

namespace redisdemo
{
    class Program
    {
        // redis config
        private static ConfigurationOptions connDCS = ConfigurationOptions.Parse("{instance_ip_address}:
{port},password=*****,connectTimeout=2000");
        //the lock for singleton
        private static readonly object Locker = new object();
        //singleton
        private static ConnectionMultiplexer redisConn;
        //singleton
        public static ConnectionMultiplexer getRedisConn()
        {
            if (redisConn == null)
            {
                lock (Locker)
                {
                    if (redisConn == null || !redisConn.IsConnected)
                    {
                        redisConn = ConnectionMultiplexer.Connect(connDCS);
                    }
                }
            }
            return redisConn;
        }
        static void Main(string[] args)
        {
            redisConn = getRedisConn();
            var db = redisConn.GetDatabase();
            //set get
            string strKey = "Hello";
            string strValue = "DCS for Redis!";
            Console.WriteLine( strKey + ", " + db.StringGet(strKey));

            Console.ReadLine();
        }
    }
}
```

*{instance\_ip\_address}* and *{port}* are the IP address/domain name and port number of the DCS Redis instance. For details about how to obtain the IP address/

domain name and port, see [Step 1](#). Change them as required. `*****` indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

**Step 7** Run the code. You have successfully accessed the instance if the following command output is displayed:

```
Hello, DCS for Redis!
```

For more information about other commands of StackExchange.Redis, visit [StackExchange.Redis](#).

----End

## 2.2.3.6 PHP

### 2.2.3.6.1 Connecting to Redis on phpredis (PHP)

This section describes how to connect to Redis on phpredis. For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

#### NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use phpredis to connect to a Redis Cluster instance, see the [phpredis description](#).

## Prerequisites

- A Redis instance is created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

## Connecting to Redis on phpredis

**Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

**Step 2** Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance through phpredis.

**Step 3** Install GCC-C++ and Make compilation components.

```
yum install gcc-c++ make
```

**Step 4** Install the PHP development package and CLI tool.

Run the following **yum** command to install the PHP development package:

**yum install php-devel php-common php-cli**

After the installation is complete, run the following command to query the PHP version and check whether the installation is successful:

**php --version****Step 5** Install the phpredis client.

1. Download the source phpredis package.

```
wget http://pecl.php.net/get/redis-5.3.7.tgz
```

This version is used as an example. To download phpredis clients of other versions, visit the Redis or PHP official website.

2. Decompress the source phpredis package.

```
tar -zxvf redis-5.3.7.tgz  
cd redis-5.3.7
```

3. Command before compilation.

```
phpize
```

4. Configure the **php-config** file.

```
./configure --with-php-config=/usr/bin/php-config
```

The location of the file varies depending on the OS and PHP installation mode. You are advised to locate the directory where the file is saved before the configuration.

```
find / -name php-config
```

5. Compile and install the phpredis client.

```
make && make install
```

6. After the installation, add the **extension** configuration in the **php.ini** file to reference the Redis module.

```
vim /etc/php.ini
```

Add the following configuration:

```
extension = "/usr/lib64/php/modules/redis.so"
```

**NOTE**

The **redis.so** file may be saved in a different directory from **php.ini**. Run the following command to locate the directory:

```
find / -name php.ini
```

7. Save the configuration and exit. Then, run the following command to check whether the extension takes effect:

```
php -m |grep redis
```

If the command output contains **redis**, the phpredis client environment has been set up.

**Step 6** Access the DCS instance by using phpredis.

1. Edit a **redis.php** file.

```
<?php  
$redis_host = "{redis_instance_address}";  
$redis_port = {port};  
$user_pwd = "{password}";  
$redis = new Redis();  
if ($redis->connect($redis_host, $redis_port) == false) {
```

```
die($redis->getLastError());
}
if ($redis->auth($user_pwd) == false) {
    die($redis->getLastError());
}
if ($redis->set("welcome", "Hello, DCS for Redis!") == false) {
    die($redis->getLastError());
}
$value = $redis->get("welcome");
echo $value;
$redis->close();
?>
```

`{redis_instance_address}` indicates the example IP address/domain name of the DCS instance and `{port}` indicates the port number of the DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change them as required. `{password}` indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. If password-free access is enabled, shield the `if` statement for password authentication.

2. Run the **php redis.php** command to access the DCS instance.

----End

### 2.2.3.6.2 Connecting to Redis on predis (PHP)

This section describes how to connect to Redis on predis. For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

#### Prerequisites

- A Redis instance is created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*
- If the ECS runs the Linux OS, ensure that the PHP compilation environment has been installed on the ECS.

#### Connecting to Redis on predis

- Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

- Step 2** Log in to the ECS.

- Step 3** Install the PHP development package and CLI tool. Run the following **yum** command:

```
yum install php-devel php-common php-cli
```

- Step 4** After the installation is complete, check the version number to ensure that the installation is successful.

```
php --version
```

**Step 5** Download the Predis package to the `/usr/share/php` directory.

1. Run the following command to download the Predis source file:

```
wget https://github.com/predis/predis/archive/refs/tags/v2.2.2.tar.gz
```

**NOTE**

This version is used as an example. To download Predis clients of other versions, visit the Redis or PHP official website.

2. Run the following commands to decompress the source Predis package:

```
tar -zxvf predis-2.2.2.tar.gz
```

3. Rename the decompressed Predis directory **predis** and move it to `/usr/share/php/`.

```
mv predis-2.2.2 predis
```

**Step 6** Edit a file used to connect to Redis.

- Example of using **redis.php** to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance:

```
<?php
require 'predis/autoload.php';
Predis\Autoloader::register();
$client = new Predis\Client([
    'scheme' => 'tcp',
    'host'   => '{redis_instance_address}',
    'port'   => {port},
    'password' => '{password}'
]);
$client->set('foo', 'bar');
$value = $client->get('foo');
echo $value;
?>
```

- Example code for using **redis-cluster.php** to connect to Redis Cluster:

```
<?php
require 'predis/autoload.php';
$servers = array(
    'tcp://{redis_instance_address}:{port}'
);
$options = array('cluster' => 'redis');
$client = new Predis\Client($servers, $options);
$client->set('foo', 'bar');
$value = $client->get('foo');
echo $value;
?>
```

`{redis_instance_address}` indicates the actual IP address/domain name of the DCS instance and `{port}` is the actual port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change them as required. `{password}` indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. If password-free access is required, delete the line that contains "password".

**Step 7** Run the `php redis.php` command to access the DCS instance.

----End

### 2.2.3.7 Connecting to Redis on ioredis (Node.js)

This section describes how to access a Redis instance on ioredis. For more information about how to use other Redis clients, visit [the Redis official website](#).

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

 NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To access a Redis Cluster instance on ioredis, see [Node.js Redis client description](#).

## Prerequisites

- A Redis instance is created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

## Connecting to Redis on ioredis

- **For client servers running Ubuntu (Debian series):**

**Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

**Step 2** Log in to the ECS.

**Step 3** Install Node.js.

```
apt install nodejs-legacy
```

If the preceding command does not work, run the following commands:

```
wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate
tar -xvf node-v4.28.5.tar.gz
cd node-v4.28.5
./configure
make
make install
```

 NOTE

After the installation is complete, run the **node --version** command to query the Node.js version to check whether the installation is successful.

**Step 4** Install the node package manager (npm).

```
apt install npm
```

**Step 5** Install the Redis client ioredis.

```
npm install ioredis
```

**Step 6** Edit the sample script for connecting to a DCS Redis instance.

Add the following content to the **ioRedisdemo.js** script, including information about connection and data reading.

```
var Redis = require('ioRedis');
var redis = new Redis({
  port: 6379, // Redis port
  host: '192.168.0.196', // Redis host
  family: 4, // 4 (IPv4) or 6 (IPv6)
  password: '*****',
  db: 0
});
```

```
redis.set('foo', 'bar');
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

*host* indicates the example IP address/domain name of the DCS instance and *port* indicates the port number of the DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change them as required. *\*\*\*\*\** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

**Step 7** Run the sample script to access the chosen DCS Redis instance.

```
node ioredisdemo.js
```

----End

- **For client servers running CentOS (Red Hat series):**

**Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see *Distributed Cache Service User Guide* > "Getting Started" > "Viewing Details of a DCS Instance".

**Step 2** Log in to the ECS.

**Step 3** Install Node.js.

```
yum install nodejs
```

If the preceding command does not work, run the following commands:

```
wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate
tar -xvf node-v0.12.4.tar.gz
cd node-v0.12.4
./configure
make
make install
```

#### NOTE

After the installation is complete, run the **node --version** command to query the Node.js version to check whether the installation is successful.

**Step 4** Install npm.

```
yum install npm
```

**Step 5** Install the Redis client ioredis.

```
npm install ioredis
```

**Step 6** Edit the sample script for connecting to a DCS Redis instance.

Add the following content to the **ioredisdemo.js** script, including information about connection and data reading.

```
var Redis = require('ioredis');
var redis = new Redis({
```



```
port: 6379, // Redis port
host: '192.168.0.196', // Redis host
family: 4, // 4 (IPv4) or 6 (IPv6)
password: '*****',
db: 0
});
redis.set('foo', 'bar');
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

*host* indicates the example IP address/domain name of the DCS instance and *port* indicates the port number of the DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change them as required. *\*\*\*\*\** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

**Step 7** Run the sample script to access the chosen DCS Redis instance.

```
node ioredisdemo.js
```

----End

## 2.2.4 Accessing a DCS Redis Instance on the Console

Access a DCS Redis instance through Web CLI. This function is supported only by DCS Redis 4.0 and later instances, and not by DCS Redis 3.0 instances.

### NOTE

- Do not enter sensitive information in Web CLI to avoid disclosure.
- Keys and values cannot contain spaces.
- If the value is empty, **nil** is returned after the **GET** command is executed.

### Prerequisites

The instance is in the **Running** state.

### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** In the row containing the desired instance, choose **More > Connect to Redis** to go to the Web CLI login page.

**Step 5** Enter the password of the DCS instance. On Web CLI, select the current Redis database, enter a Redis command in the command box, and press **Enter**.

 **NOTE**

If no operation is performed for more than 5 minutes, the connection times out. You must enter the access password to connect to the instance again.


----End

## 2.3 Viewing Details of a DCS Instance

On the DCS console, you can view DCS instance details.

### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner of the console and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.


**Step 4** Search for DCS instances using any of the following methods:




- Search by keyword.  
If you do not select an attribute and enter a keyword in the search box, the system searches for instances by instance name by default.
- Select attributes and enter their keywords to search.  
Click the search box, select an instance attribute, and enter a keyword to search. You can select multiple attributes at a time.  
For example, choose **Status > Running**, **Type > Master/Standby**, and **Cache Engine > Redis 5.0**.


For more information on how to search, click the question mark to the right of the search box.

**Step 5** On the DCS instance list, click the name of a DCS instance to display more details about it. The following table describes the parameters.

**Table 2-17** Parameters on the Basic Information page of a DCS instance

Section	Parameter	Description
Instance Details	Name	Name of the DCS instance. To modify the instance name, click  .
	Status	State of the chosen instance.
	ID	ID of the chosen instance.
	Cache Engine	Cache engine and cache engine version used by the DCS instance. For example, Redis 5.0.
	Instance Type	Type of the selected instance. Currently, supported types include single-node, master/standby, Proxy Cluster, and Redis Cluster.

Section	Parameter	Description
	Cache Size	Specification of the chosen instance.
	Used/ Available Memory (MB)	The used memory space and maximum available memory space of the chosen instance. The used memory space includes: <ul style="list-style-type: none"> <li>• Size of data stored on the DCS instance</li> <li>• Size of Redis-server buffers (including client buffer and repl-backlog) and internal data structures</li> </ul>
	CPU	CPU of the DCS instance.
	Created	Time at which the chosen instance started to be created.
	Run	Time at which the instance was created.
	Maintenance	Time range for any scheduled maintenance activities on cache nodes of this DCS instance. To modify the time window, click the  icon.
	Description	Description of the DCS instance. To modify the description, click  .
Connection	Password Protected	Currently, password-protected access and password-free access are supported.
	Connection Address	Domain name and port number of the instance. You can click  next to <b>Connection Address</b> to change the port. <b>NOTE</b> <ul style="list-style-type: none"> <li>• The instance has a domain name address if it was created after DCS has been interconnected with DNS. Instances created before the interconnection only have IP addresses and cannot be changed to domain name access.</li> <li>• For a master/standby DCS Redis 4.0 and later instance, <b>Connection Address</b> indicates the domain name and port number of the master node, and <b>Read-only Address</b> indicates the domain name and port number of the standby node. When connecting to such an instance, you can use the domain name and port number of the master node or the standby node.</li> <li>• You can change the port only for a DCS Redis 4.0 and later instance, but not for a DCS Redis 3.0 instance.</li> </ul>
	IP Address	IP address and port number of the chosen instance.
Network	AZ	Availability zone in which the cache node running the selected DCS instance resides.
	VPC	VPC in which the chosen instance resides.

Section	Parameter	Description
	Subnet	Subnet in which the chosen instance resides.
	Security Group	Security group that controls access to the DCS instance. Only Redis 3.0 supports access control with security groups. To modify the security group, click  . DCS Redis 4.0 and later are based on VPC Endpoint and do not support security groups. You can click <b>configure the whitelist</b> to configure the whitelist.
Instance Topology	-	Hover the mouse pointer over an instance to view its metrics, or click the icon of an instance to view its historical metrics. Topologies are supported only for master/standby and cluster instances.

----End

# 3 User Guide

---

## 3.1 Permissions Management

### 3.1.1 Creating a User and Granting DCS Permissions

This chapter describes how to use IAM to implement fine-grained permissions control for your DCS resources. With IAM, you can:

- Create IAM users for employees based on your enterprise's organizational structure. Each IAM user will have their own security credentials for accessing DCS resources.
- Grant only the permissions required for users to perform a specific task.
- Entrust an account or cloud service to perform efficient O&M on your DCS resources.

If your account does not need individual IAM users, you may skip over this chapter.

This section describes the procedure for granting the **DCS ReadOnlyAccess** permission (see [Figure 3-1](#)) as an example.

#### Prerequisites

You are familiar with the permissions (see [Permissions Management](#)) supported by DCS and choose policies or roles according to your requirements. For the permissions of other services, see "Permissions".

## Process Flow

**Figure 3-1** Process of granting DCS permissions



1. Create a user group and assign permissions to it.  
Create a user group on the IAM console, and attach the **DCS ReadOnlyAccess** policy to the group.
2. Create an IAM user.  
Create a user on the IAM console and add the user to the group created in **1**.
3. Log in and verify permissions.  
Log in to the DCS console by using the newly created user, and verify that the user only has read permissions for DCS.

### 3.1.2 DCS Custom Policies

Custom policies can be created to supplement the system-defined policies of DCS. For the actions that can be added to custom policies, see section "Permissions Policies and Supported Actions" in the *Distributed Cache Service API Reference*.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see section "Creating a Custom Policy" in the *Identity and Access Management API Reference*. The following section contains examples of common DCS custom policies.

 NOTE

Due to data caching, a policy involving OBS actions will take effect five minutes after it is attached to a user, user group, or project.

## Example Custom Policies

- Example 1: Allowing users to delete and restart DCS instances and clear data of an instance

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dcs:instance:delete",
        "dcs:instance:modifyStatus"
      ]
    }
  ]
}
```

- Example 2: Denying DCS instance deletion

A policy with only "Deny" permissions must be used in conjunction with other policies to take effect. If the permissions assigned to a user contain both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

The following method can be used if you need to assign permissions of the **DCS FullAccess** policy to a user but you want to prevent the user from deleting DCS instances. Create a custom policy for denying DCS instance deletion, and attach both policies to the group to which the user belongs. Then, the user can perform all operations on DCS instances except deleting DCS instances. The following is an example of a deny policy:

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "dcs:instance:delete"
      ]
    }
  ]
}
```

## 3.2 Operating DCS Instances

### 3.2.1 Modifying DCS Instance Specifications

On the DCS console, you can scale a DCS Redis instance to a larger or smaller capacity.

 NOTE

- **Modify instance specifications during off-peak hours.** If the modification failed in peak hours (for example, when memory or CPU usage is over 90% or write traffic surges), try again during off-peak hours.
- You can only change the instance type of single-node or master/standby DCS Redis 3.0 instances.
- If your DCS instances are too old to support scaling, contact technical support to upgrade the instances.
- Services may be interrupted for seconds during the modification. Therefore, services connected to Redis must support reconnection.
- Modifying instance specifications does not affect the connection address, password, data, security group, and whitelist configurations of the instance.

## Change of the Instance Type

- **Supported instance type changes:**
  - From single-node to master/standby: Supported by Redis 3.0, and not by Redis 4.0/5.0.
  - From master/standby to Proxy Cluster: Supported by Redis 3.0, and not by Redis 4.0/5.0/6.0.

If the data of a master/standby DCS Redis 3.0 instance is stored in multiple databases, or in non-DB0 databases, the instance cannot be changed to the Proxy Cluster type. A master/standby instance can be changed to the Proxy Cluster type only if its data is stored only on DB0.

  - From cluster types to other types: Not supported.
- **Impact of instance type changes:**
  - From single-node to master/standby for a DCS Redis 3.0 instance:  
The instance cannot be connected for several seconds and remains read-only for about 1 minute.
  - From master/standby to Proxy Cluster for a DCS Redis 3.0 instance:  
The instance cannot be connected and remains read-only for 5 to 30 minutes.

## Scaling

- **The following table lists scaling options supported by different DCS instances.**

**Table 3-1** Scaling options supported by different DCS instances

Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster
Redis 3.0	Scaling up/down	Scaling up/down	N/A	Scaling out



Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster
Redis 4.0/5.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/down, out/in, and replica quantity change	Scaling up/down and out/in
Redis 6.0	Scaling up/down	Scaling up/down	Scaling up/down, out/in, and replica quantity change	N/A

 **NOTE**

If the reserved memory of a DCS Redis 3.0 instance is insufficient, the scaling may fail when the memory is used up.

Change the replica quantity and capacity separately.


- **Impact of scaling:**

- Single-node and master/standby
  - A DCS Redis 4.0 or later instance will be disconnected for several seconds and remain read-only for about 1 minute.
  - A DCS Redis 3.0 instance will be disconnected and remain read-only for 5 to 30 minutes.
  - For scaling up, only the memory of the instance is expanded. The CPU processing capability is not improved.
  - Data of single-node instances may be lost because they do not support data persistence. After scaling, check whether the data is complete and import data if required.
  - Backup records of master/standby instances cannot be used after scaling down.
- Cluster
  - If the shard quantity is not decreased, the instance can always be connected, but the CPU usage will increase, compromising performance by up to 20%, and the latency will increase during data migration.
  - During scaling up, new Redis Server nodes are added, and data is automatically balanced to the new nodes.
  - Nodes will be deleted if the shard quantity decreases. To prevent disconnection, ensure that the deleted nodes are not directly referenced in your application.

- Ensure that the used memory of each node is less than 70% of the maximum memory per node of the new flavor. Otherwise, you cannot perform the scale-in.
  - If the memory becomes full during scaling due to a large amount of data being written, scaling will fail. Modify specifications during off-peak hours.
  - Scaling involves data migration. The latency for accessing the key being migrated increases. For a Redis Cluster instance, ensure that the client can properly process the **MOVED** and **ASK** commands. Otherwise, requests will fail.
  - Backup records created before scaling cannot be restored.
- **Notes on changing the number of replicas of a DCS Redis instance:**  
Deleting replicas interrupts connections. If your application cannot reconnect to Redis or handle exceptions, you need to restart the application after scaling.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Choose **More > Modify Specifications** in the row containing the DCS instance.

**Step 5** On the **Modify Specifications** page, select the desired specification.

 **NOTE**

For a master/standby or Redis Cluster DCS Redis 4.0 or later instance, you can choose to change by specification or replica quantity.

**Step 6** Set **Apply Change** to **Now** or **During maintenance**.

Select **During maintenance** if the modification interrupts connections.

**Table 3-2** Scenarios where specification modification interrupts connections

Change	When Connections Are Interrupted
Scaling up a single-node or master/standby instance	Memory is increased from a size smaller than 8 GB to 8 GB or larger.
Scaling down a Proxy Cluster and Redis Cluster instance	The number of shards is decreased.
Deleting replicas	Replicas are deleted from a master/standby or Redis Cluster instance.

 **NOTE**

- If the modification does not interrupt connections, it will be applied immediately even if you select **During maintenance**.
- The modification cannot be withdrawn once submitted. To reschedule a modification, you can change the maintenance window. The maintenance window can be changed up to three times.
- Modifications on DCS Redis 3.0 instances can only be applied immediately.
- If you apply the change during maintenance, the change starts at any time within the maintenance window, rather than at the start time of the window.
- If a large amount of data needs to be migrated when you scale down a cluster instance, the operation may not be completed within the maintenance window.

**Step 7** Click **Next**. In the dialog box that is displayed, click **Yes**.

**Step 8** Click **Submit** to start modifying the DCS instance.

You can go to **Background Tasks** page to view the modification status. For more information, see [Viewing Background Tasks](#).

Specification modification of a single-node or master/standby DCS instance takes about 5 to 30 minutes to complete, while that of a cluster DCS instance takes a longer time. After an instance is successfully modified, it changes to the **Running** state.

 **NOTE**

- If the specification modification of a single-node DCS instance fails, the instance is temporarily unavailable for use. The specification remains unchanged. Some management operations (such as parameter configuration and specification modification) are temporarily not supported. After the specification modification is completed in the backend, the instance changes to the new specification and becomes available for use again.
- If the specification modification of a master/standby or cluster DCS instance fails, the instance is still available for use with its original specifications. Some management operations (such as parameter configuration, backup, restoration, and specification modification) are temporarily not supported. Remember not to read or write more data than allowed by the original specifications; otherwise, data loss may occur.
- After the specification modification is successful, the new specification of the instance takes effect.
- If the error "DCS.4104 The DCS instance is recovering from an internal fault. Please try again." is displayed, contact O&M personnel.

----End

## 3.2.2 Restarting DCS Instances

On the DCS console, you can restart one or multiple DCS instances at a time.

#### NOTICE


- After a single-node DCS instance is restarted, data will be deleted from the instance.
- While a DCS instance is restarting, it cannot be read from or written to.
- An attempt to restart a DCS instance while it is being backed up may result in a failure.
- Restarting a DCS instance will disconnect the original client. You are advised to configure automatic reconnection in your application.

## Prerequisites

The DCS instances you want to restart are in the **Running** or **Faulty** state.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** On the **Cache Manager** page, select one or more DCS instances you want to restart.

**Step 5** Click **Restart** above the DCS instance list.

**Step 6** In the displayed dialog box, click **Yes**.

It takes 10 seconds to 30 minutes to restart DCS instances. After DCS instances are restarted, their status changes to **Running**.

#### NOTE

- To restart a single instance, you can also click **Restart** in the same row as the instance.
- The time required for restarting a DCS instance depends on the cache size of the instance.

----End

## 3.2.3 Deleting DCS Instances

On the DCS console, you can delete one or multiple DCS instances at a time. You can also delete all instance creation tasks that have failed to run.

#### NOTICE

- After a DCS instance is deleted, the instance data will also be deleted without backup. In addition, any backup data of the instance will be deleted. Therefore, download the backup files of the instance for permanent storage before deleting the instance.
- If the instance is in cluster mode, all cluster nodes will be deleted.

## Prerequisites

- The DCS instances you want to delete have been created.
- The DCS instances you want to delete are in the **Running** or **Faulty** state.

## Procedure

Deleting DCS Instances

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** On the **Cache Manager** page, select one or more DCS instances you want to delete.

DCS instances in the **Creating**, **Restarting**, **Upgrading**, **Resizing**, **Clearing data**, **Backing up**, or **Restoring** state cannot be deleted.

**Step 5** Choose **More > Delete** above the instance list.

**Step 6** Enter **DELETE** as prompted and click **Yes** to delete the instance.

It takes 1 to 30 minutes to delete DCS instances.

#### NOTE

To delete a single instance, choose **Operation > More > Delete** in the same row as the instance.

----End

Deleting Instance Creation Tasks That Have Failed to Run

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** If there are DCS instances that have failed to be created, **Instance Creation Failures** is displayed above the instance list.

**Step 5** Click the icon or the number of failed tasks next to **Instance Creation Failures**.

The **Instance Creation Failures** dialog box is displayed.

**Step 6** Choose failed instance creation tasks to delete.

- To delete all failed tasks, click **Delete All** above the task list.
- To delete a single failed task, click **Delete** in the same row as the task.

----End

## 3.2.4 Performing a Master/Standby Switchover for a DCS Instance

On the DCS console, you can manually switch the master and standby nodes of a DCS instance. This operation is used for special purposes, for example, releasing all service connections or terminating ongoing service operations.

Only master/standby instances support a master/standby node switchover.

### NOTICE

- Services may be interrupted for up to 10 seconds during the switchover. Before performing a switchover, ensure that your application supports reconnection.
- During a master/standby node switchover, a large amount of resources will be consumed for data synchronization between the master and standby nodes. You are advised to perform this operation during off-peak hours.
- Data of the maser and standby nodes is synchronized asynchronously. Therefore, a small amount of data that is being operated on during the switchover may be lost.

## Prerequisites

The DCS instance for which you want to perform a master/standby node switchover is in the **Running** state.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** In the **Operation** column of the instance, choose **More > Master/Standby Switchover**, then click **OK**.

----End

## 3.2.5 Clearing DCS Instance Data

You can clear data of DCS Redis 4.0 and later instances on the console.

**Clearing instance data cannot be undone and cleared data cannot be recovered. Exercise caution when performing this operation.**

## Prerequisites

The instance is in the **Running** state.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Select one or more DCS instances.

**Step 5** Click **Clear data** above the instance list.

**Step 6** In the displayed dialog box, click **Yes**.

----End

## 3.2.6 Exporting DCS Instance List

On the DCS console, you can export DCS instance information in full to an Excel file.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click **Export** above the instance list to export the DCS instance list.

----End

## 3.3 Managing DCS Instances

### 3.3.1 Configuration Notice

- In most cases, different DCS instance management operations cannot proceed concurrently. If you initiate a new management operation while the current operation is in progress, the DCS console prompts you to initiate the new operation again after the current operation is complete. DCS instance management operations include:
  - Creating a DCS instance
  - Configuring parameters
  - Restarting a DCS instance
  - Changing the instance password
  - Resetting the instance password

- Scaling, backing up, or restoring an instance
- You can restart a DCS instance while it is being backed up, but the backup task will be forcibly interrupted and is likely to result in a backup failure.

---

#### NOTICE

In the event that a cache node of a DCS instance is faulty:

- The instance remains in the **Running** state and you can continue to read from and write to the instance. This is achieved thanks to the high availability of DCS.
  - Cache nodes can recover from internal faults automatically. Manual fault recovery is also supported.
  - Certain operations (such as parameter configuration, password change or resetting, backup, restoration, and specification modification) in the management zone are not supported during fault recovery. You can contact technical support or perform these operations after the cache nodes recover from faults.
- 

### 3.3.2 Modifying Configuration Parameters


You can modify the configuration parameters of your DCS instance to optimize DCS performance based on your requirements.

For example, if you do not need data persistence, set **appendonly** to **no**.

#### NOTE

After the instance configuration parameters are modified, the modification takes effect immediately without the need to manually restart the instance. For a cluster instance, the modification takes effect on all shards.

#### Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** On the **Cache Manager** page, click the name of the DCS instance you want to configure.
- Step 5** Choose **Instance Configuration > Parameters**.
- Step 6** On the **Parameters** page, click **Modify**.
- Step 7** Modify parameters based on your requirements.

[Table 3-3](#) describes the parameters. In most cases, retain the default values.



**Table 3-3** DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
active-expire-num	Number of expired keys that can be deleted in regular scans. Redis 3.0 instances do not have this parameter.	1-1000	20
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of <b>0</b> means that this function is disabled. Proxy Cluster instances do not have this parameter.	0-7200 seconds	0
appendfsync	Controls how often fsync() transfers cached data to the disk. Note that some OSs will perform a complete data transfer but some others only make a "best-effort" attempt. There are three settings: no: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance. always: fsync() is called after every write to the AOF. This mode is very slow, but also very safe. everysec: fsync() is called once per second. This mode provides a compromise between safety and performance. Single-node instances do not have this parameter.	<ul style="list-style-type: none"> <li>• no</li> <li>• always</li> <li>• everysec</li> </ul>	no

Parameter	Description	Value Range	Default Value
appendonly	Indicates whether to log each modification of the instance (that is, data persistence). By default, data is written to disks asynchronously in Redis. If this function is disabled, recently-generated data might be lost in the event of a power failure. Options: yes: enabled no: disabled  Single-node instances do not have this parameter.	<ul style="list-style-type: none"> <li>• yes</li> <li>• no</li> </ul>	yes
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above <b>client-output-buffer-slave-soft-limit</b> before the client is disconnected.  Single-node instances do not have this parameter.	0-60	60
client-output-buffer-slave-hard-limit	Hard limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the hard limit, the client is immediately disconnected.  Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
client-output-buffer-slave-soft-limit	Soft limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the soft limit and continuously remains above the limit for the time specified by the <b>client-output-buffer-limit-slave-soft-seconds</b> parameter, the client is disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
maxmemory-policy	<p>The deletion policy to apply when the maxmemory limit is reached. Options:</p> <p><b>volatile-lru:</b> Evict keys by trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set. <b>(Recommended)</b></p> <p><b>allkeys-lru:</b> Evict keys by trying to remove the LRU keys first.</p> <p><b>volatile-random:</b> evict keys randomly, but only evict keys with an expire set.</p> <p><b>allkeys-random:</b> Evict keys randomly.</p> <p><b>volatile-ttl:</b> Evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first.</p> <p><b>noeviction:</b> Do not delete any keys and only return errors when the memory limit was reached.</p> <p><b>volatile-lfu:</b> Evict keys by trying to remove the less frequently used (LFU) keys first, but only among keys that have an expire set.</p> <p><b>allkeys-lfu:</b> Evict keys by trying to remove the LFU keys first.</p>	Depends on the instance version.	Depends on the instance version and type.
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	100-5000	5,000

Parameter	Description	Value Range	Default Value
master-read-only	Sets the instance to be read-only. All write operations will fail. Proxy Cluster instances do not have this parameter.	<ul style="list-style-type: none"> <li>yes</li> <li>no</li> </ul>	no
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance. Proxy Cluster instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
proto-max-bulk-len	Maximum size of a single element request (in bytes).	1,048,576–536,870,912	536,870,912
repl-backlog-size	The replication backlog size (bytes). The backlog is a buffer that accumulates replica data when replicas are disconnected from the master. When a replica reconnects, a partial synchronization is performed to synchronize the data that was missed while replicas were disconnected. Single-node instances do not have this parameter.	16,384–1,073,741,824	1,048,576
repl-backlog-ttl	The amount of time, in seconds, before the backlog buffer is released, starting from the last a replica was disconnected. The value <b>0</b> indicates that the backlog is never released. Single-node instances do not have this parameter.	0–604,800	3,600

Parameter	Description	Value Range	Default Value
repl-timeout	Replication timeout (in seconds). Single-node instances do not have this parameter.	30-3,600	60
hash-max-ziplist-entries	Hashes are encoded using a memory efficient data structure when the number of entries in hashes is less than the value of this parameter.	1-10,000	512
hash-max-ziplist-value	Hashes are encoded using a memory efficient data structure when the biggest entry in hashes does not exceed the length threshold indicated by this parameter.	1-10,000	64
set-max-intset-entries	When a set is composed of just strings that happen to be integers in radix 10 in the range of 64 bit signed integers, sets are encoded using a memory efficient data structure.	1-10,000	512
zset-max-ziplist-entries	Sorted sets are encoded using a memory efficient data structure when the number of entries in sorted sets is less than the value of this parameter.	1-10,000	128
zset-max-ziplist-value	Sorted sets are encoded using a memory efficient data structure when the biggest entry in sorted sets does not exceed the length threshold indicated by this parameter.	1-10,000	64

Parameter	Description	Value Range	Default Value
latency-monitor-threshold	<p>Threshold time in latency monitoring. Unit: millisecond.</p> <p>Set to <b>0</b>: Latency monitoring is disabled.</p> <p>Set to more than 0: All with at least this many milliseconds of latency will be logged.</p> <p>By running the <b>LATENCY</b> command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring.</p> <p>Proxy Cluster instances do not have this parameter.</p>	0-86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	<p>Controls which keyspace events notifications are enabled for. If the value is an empty string, this function is disabled. A combination of different values can be used to enable notifications for multiple event types. Possible values:</p> <p><b>K</b>: Keyspace events, published with the <b>__keyspace@__</b> prefix.</p> <p><b>E</b>: Keyevent events, published with <b>__keyevent@__</b> prefix</p> <p><b>g</b>: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME</p> <p><b>\$</b>: String commands</p> <p><b>l</b>: List commands</p> <p><b>s</b>: Set commands</p> <p><b>h</b>: Hash commands</p> <p><b>z</b>: Sorted set commands</p> <p><b>x</b>: Expired events (events generated every time a key expires)</p> <p><b>e</b>: Evicted events (events generated when a key is evicted for maxmemory)</p> <p><b>A</b>: an alias for "g\$lshzxe"</p> <p>The parameter value must contain either <b>K</b> or <b>E</b>. <b>A</b> cannot be used together with any of the characters in "g\$lshzxe". For example, the value <b>Kl</b> means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value <b>AKE</b> means Redis will notify Pub/Sub clients about all events.</p>	See the parameter description.	Ex



Parameter	Description	Value Range	Default Value
	Proxy Cluster instances do not have this parameter.		
slowlog-log-slower-than	Redis records queries that exceed a specified execution time. <b>slowlog-log-slower-than</b> is the maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will record the query.	1000–1,000,000	10,000
slowlog-max-len	The maximum allowed number of slow queries that can be logged. Slow query log consumes memory, but you can reclaim this memory by running the <b>SLOWLOG RESET</b> command.	0–1000	128

 NOTE

- For more information about the parameters described in [Table 3-3](#), visit <https://redis.io/topics/memory-optimization>.
- The **latency-monitor-threshold** parameter is usually used for fault location. After locating faults based on the latency information collected, change the value of **latency-monitor-threshold** to **0** to avoid unnecessary latency.
- More about the **notify-keyspace-events** parameter:
  - The parameter setting must contain at least a **K** or **E**.
  - A** is an alias for "g\$lshzxe" and cannot be used together with any of the characters in "g\$lshzxe".
  - For example, the value **KI** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.
- Configurable parameters and their values vary depending on the instance type.

**Step 8** After you have finished setting the parameters, click **Save**.

**Step 9** Click **Yes** to confirm the modification.

 **NOTE**

If the error "111400104 The DCS instance is recovering from an internal fault. Please try again." is displayed, do not modify the configuration parameters. Provide the error information to O&M personnel.

----End

## Typical Scenarios of Configuring Parameters

The following describes how to change the value of the **appendonly** parameter:

- If Redis is used as the cache and services are insensitive to Redis data losses, disable instance persistence to improve performance. In this case, change the value of **appendonly** to **no**. For details, see [Procedure](#).
- If Redis is used as the database or services are sensitive to Redis data losses, enable instance persistence. In this case, change the value of **appendonly** to **yes**. For details, see [Procedure](#). After instance persistence is enabled, you need to consider the frequency of writing Redis cache data to disks and the impact on the Redis performance. You can use this parameter together with the **appendfsync** parameter. There are three modes of calling `fsync()`:
  - **no**: `fsync()` is never called. The OS will flush data when it is ready. This mode offers the highest performance.
  - **always**: `fsync()` is called after every write to the AOF. This mode is very slow, but also very safe.
  - **everysec**: `fsync()` is called once per second, ensuring both data security and performance.

 **NOTE**

Currently, the **appendonly** and **appendfsync** parameters can be modified on the console only for master/standby and Redis 4.0 and later Redis Cluster instances.

### 3.3.3 Modifying Maintenance Time Window

On the DCS console, after creating a DCS instance, you can modify the maintenance time window of the DCS instance on the instance's **Basic Information** page.

#### Prerequisites

At least one DCS instance has been created.




#### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click the name of the DCS instance for which you want to modify the maintenance time window.

- Step 5** Click the **Basic Information** tab. In the **Instance Details** area, click the  icon next to the **Maintenance** parameter.
- Step 6** Select a new maintenance time window from the drop-down list. Click  to save the modification or  to discard the modification.

The modification will take effect immediately, that is, the new maintenance time window will appear on the **Basic Information** tab page immediately.

----End

### 3.3.4 Modifying the Security Group





On the DCS console, after creating a DCS instance, you can modify the security group of the DCS instance on the instance's **Basic Information** page.

You can modify the security groups of DCS Redis 3.0 instances but cannot modify those of DCS Redis 4.0/5.0/6.0 instances.

#### Prerequisites

At least one DCS instance has been created.

#### Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance for which you want to modify the security group.
- Step 5** Click the **Basic Information** tab. In the **Network** area, click  next to the **Security Group** parameter.
- Step 6** Select a new security group from the drop-down list. Click  to save the modification or  to discard the modification.

#### NOTE

Only the security groups that have been created can be selected from the drop-down list. If you need to create a security group, follow the procedure described [Security Group Configurations](#).

Security group rules must meet the following requirements:

**Type:** IPv4; **Protocol:** TCP/any; **Port:** 6379/11211; **IP address:** all CIDR blocks/a security group ID/a CIDR block

The modification will take effect immediately, that is, the new maintenance time window will appear on the **Basic Information** tab page immediately.

----End

### 3.3.5 Viewing Background Tasks

After you initiate certain instance operations such as modifying instance specifications and changing or resetting a password, a background task will start for the operation. On the DCS console, you can view the background task status and clear task information by deleting task records.

#### Procedure

**Step 1** Log in to the DCS console.


**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click the name of the desired instance on the left.

**Step 5** Choose **Background Tasks**.

Filter tasks by specifying the time, property, or keyword.

- Click  to refresh the task status.
- To clear the record of a background task, click **Delete** in the **Operation** column.

#### NOTE

You can only delete the records of tasks in the **Successful** or **Failed** state.

----End

### 3.3.6 Viewing Data Storage Statistics of a DCS Redis 3.0 Proxy Cluster Instance

You can view the data storage statistics of all nodes of a DCS Redis 3.0 Proxy Cluster instance. If data storage is unevenly distributed across nodes, you can scale up the instance or clear data.

You can only view data storage statistics of DCS Redis 3.0 Proxy Cluster instances. Instances of other types, for example, master/standby, only have one node, and you can view the used memory on the instance details page.

#### NOTE

A Redis Cluster instance has multiple storage nodes. You can check the data storage statistics of a Redis Cluster instance in its Redis Server monitoring data.

DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. You are advised to use DCS Redis 4.0 and later instances.

#### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

Filter DCS instances to find the desired one.

**Step 4** Click the name of the desired Proxy Cluster instance to go to the instance details page.

**Step 5** Click the **Node Management** tab.

The data volume of each node in the cluster instance is displayed.

When the data storage capacity of a node in a cluster is used up, you can scale up the instance according to [Modifying DCS Instance Specifications](#).

----End

### 3.3.7 Managing Shards and Replicas

This section describes how to query the shards and replicas of a DCS Redis 4.0 or later instance and how to manually promote a replica to master.

Currently, **this function is supported only by master/standby and cluster DCS Redis 4.0 and later instances. DCS Redis 3.0 instances and single-node instances do not support this function.**

- A master/standby instance has only one shard with one master and one replica by default. You can view the sharding information on the **Shards and Replicas** page. To manually switch the master and replica roles, see [Performing a Master/Standby Switchover for a DCS Instance](#).
- A Proxy Cluster instance has multiple shards. Each shard has one master and one replica by default. On the **Shards and Replicas** page, you can view the sharding information. For details about the number of shards corresponding to different instance specifications, see [Proxy Cluster DCS Redis 4.0 and 5.0 Instances](#).
- A Redis Cluster instance has multiple shards. Each shard has one master and one replica by default. On the **Shards and Replicas** page, you can view the sharding information. For details about the number of shards corresponding to different instance specifications, see [Redis Cluster](#).

#### Promoting a Replica to Master

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**. The **Cache Manager** page is displayed.

**Step 4** Click an instance.

**Step 5** Choose **Shards and Replicas**.

The page displays all shards in the instance and the list of replicas of each shard.

**Step 6** Click  to show all replicas of a shard.

- **Failover Priority** can be set for replicas for a master/standby instance. **Remove IP Address** is supported in master/standby instances.

Failover priority: If the master fails, the system will be automatically promoted to the replica with the highest priority you specified. If there are multiple replicas sharing the same priority, a selection and switch process will be performed in the internal system. Priority ranges from 0 to 100 in descending order. **0** indicates that the replica will never be automatically promoted, **1** indicates the highest priority, and **100** indicates the lowest priority.

Remove IP address: If a master/standby instance has more than one replicas (excluding master), click **Remove IP Address** to remove the IP addresses of extra replicas. After a replica IP address is removed, the read-only domain name will no longer be resolved to the replica IP address. If a master/standby instance has only one replica, its IP address cannot be removed.

----End

### 3.3.8 Analyzing Big Keys and Hot Keys

By performing big key analysis and hot key analysis, you will have a picture of keys that occupy a large space and keys that are the most frequently accessed.

#### Notes on big key analysis:

- All DCS Redis instances support big key analysis.
- During big key analysis, all keys will be traversed. The larger the number of keys, the longer the analysis takes.
- Perform big key analysis during off-peak hours and avoid automatic backup periods.
- For a master/standby or cluster instance, the big key analysis is performed on the standby node, so the impact on the instance is minor. For a single-node instance, the big key analysis is performed on the only node of the instance and will reduce the instance access performance by up to 10%. Therefore, perform big key analysis on single-node instances during off-peak hours.
- A maximum of 100 big key analysis records (20 for Strings; 80 for Lists/Sets/Zsets/Hashes and max. 20 for each type) are retained for each instance. When this limit is reached, the oldest records will be deleted to make room for new records. You can also manually delete records you no longer need.


#### Notes on hot key analysis:

- Supported only by DCS Redis 4.0 and later instances, and the **maxmemory-policy** parameter of the instances must be set to **allkeys-lfu** or **volatile-lfu**.
- During hot key analysis, all keys will be traversed. The larger the number of keys, the longer the analysis takes.
- Perform hot key analysis shortly after peak hours to ensure the accuracy of the analysis results.
- The hot key analysis is performed on the master node of each instance and will reduce the instance access performance by up to 10%.
- A maximum of 100 hot key analysis records are retained for each instance. When this limit is reached, the oldest records will be deleted to make room for new records. You can also manually delete records you no longer need.

 **NOTE**

Perform big key and hot key analysis during off-peak hours to avoid 100% CPU usage.

## Big Key Analysis Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of a DCS Redis instance.
- Step 5** Choose **Analysis and Diagnosis > Cache Analysis**.
- Step 6** On the **Big Key Analysis** tab page, manually perform big key analysis or schedule daily automatic analysis.
- Step 7** After an analysis task completes, click **View** to view the analysis results.

You can view the analysis results of different data types.

 **NOTE**


A maximum of 20 big key analysis records are retained for Strings and 80 are retained for Lists, Sets, Zsets, and Hashes.

**Table 3-4** Results of big key analysis

Parameter	Description
Key	Name of a big key.
Type	Type of a big key, which can be string, list, set, zset, or hash.
Size	Size or number of elements of a big key.
Shard	Shard where the big key of the cluster instance is located.
Database	Database where a big key is located.

----End

## Hot Key Analysis Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of a DCS Redis instance.

**Step 5** Choose **Analysis and Diagnosis > Cache Analysis**.

**Step 6** On the **Hot Key Analysis** tab page, manually perform hot key analysis or schedule daily automatic analysis.

 **NOTE**

If hot key analysis cannot be performed, set **maxmemory-policy** to **allkeys-lfu** or **volatile-lfu**. If this parameter has already been set to **allkeys-lfu** or **volatile-lfu**, perform hot key analysis right away.

**Step 7** After an analysis task completes, click **View** to view the analysis results.

The hot key analysis results are displayed.

 **NOTE**

The console displays a maximum of 100 hot key analysis records for each instance.

**Table 3-5** Results of hot key analysis

Parameter	Description
Key	Name of a hot key.
Type	Type of a hot key, which can be string, hash, list, set, or sorted set.
Size	Size of the hot key value.
FREQ	Reflects the access frequency of a key within a specific period of time. <b>FREQ</b> is the logarithmic access frequency counter. The maximum value of <b>FREQ</b> is 255, which indicates 1 million access requests. After <b>FREQ</b> reaches 255, it will no longer increment even if access requests continue to increase. <b>FREQ</b> will decrement by 1 for every minute during which the key is not accessed.
Shard	Shard where the hot key of the cluster instance is located.
DataBase	Database where a hot key is located.

----End

### 3.3.9 Scanning and Deleting Expired Keys in a DCS Redis Instance

There are two ways to delete a key in Redis.

- Use the **DEL** command to directly delete a key.
- Use commands such as **EXPIRE** to set a timeout on a key. After the timeout elapses, the key becomes inaccessible but is not deleted immediately because Redis is mostly single-threaded. Redis uses the following strategies to release the memory used by expired keys:



- Lazy free deletion: The deletion strategy is controlled in the main I/O event loop. Before a read/write command is executed, a function is called to check whether the key to be accessed has expired. If it has expired, it will be deleted and a response will be returned indicating that the key does not exist. If the key has not expired, the command execution resumes.
- Scheduled deletion: A time event function is executed at certain intervals. Each time the function is executed, a random collection of keys are checked, and expired keys are deleted. Instead of checking all keys each time, open-source Redis randomly checks 20 keys each time, 10 times per second by default. This avoids prolonging blocks on the Redis main thread, but the memory used by expired keys cannot be released quickly.

DCS integrates these strategies, and provides a common expired key query method to allow you to periodically release the memory used by expired keys. You can configure scheduled scans on the master nodes of your instances. The entire keyspace is traversed during the scans, triggering Redis to check whether the keys have expired and to remove expired keys if any.

 **NOTE**

Perform expired key scans during off-peak hours to avoid 100% CPU usage.


## Notes and Constraints

Expired keys can be scanned only for DCS Redis 4.0 and later instances.

Released expired keys cannot be queried.

## Scanning and Deleting Expired Keys in a DCS Redis Instance

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner of the management console and select the region where your instance is located.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click the name of a DCS Redis instance.

**Step 5** Choose **Analysis and Diagnosis > Cache Analysis**.

**Step 6** On the **Expired Key Scan** tab page, scan for expired keys and release them.

- Click **Start Scanning** to scan for expired keys immediately.
- Enable **Scheduled** to schedule automatic scans at a specified time. For details about how to configure automatic scans, see [Table 3-6](#) and [Automated Scan Performance and Suggestions](#).

**Table 3-6** Parameters for scheduling automatic scans

Parameter	Description
Start At	The first scan can only start after the current time. Format: MM/DD/YYYY hh:mm:ss

Parameter	Description
Interval	<p>Interval between scans.</p> <ul style="list-style-type: none"> <li>If the previous scan is not complete when the start time arrives, the upcoming scan will be skipped.</li> <li>If the previous scan is complete within five minutes after the start time, the upcoming scan will not be skipped.</li> </ul> <p><b>NOTE</b> Continuous scans may cause high CPU usage. Set this parameter based on the total number of keys in the instance and the increase of keys. For details, see <a href="#">Automated Scan Performance and Suggestions</a>.</p> <p>Value range: 0–43,200 Default value: 1440 Unit: minute</p>
Timeout	<p>This parameter is used to prevent scanning timeout due to unknown reasons. If scanning times out due to unknown reasons, subsequent scheduled tasks cannot be executed. After the specified timeout elapses, a failure message is returned and the next scan will be performed.</p> <ul style="list-style-type: none"> <li>Set the timeout to at least twice the interval.</li> <li>You can set a value based on the time taken in previous scans and the maximum timeout that can be tolerated in the application scenario.</li> </ul> <p>Value range: 1–86,400 Default value: 2880 Unit: minute</p>
Keys to Iterate	<p>The <b>SCAN</b> command is used to iterate the keys in the current database. The <b>COUNT</b> option is used to let the user tell the iteration command how many elements should be returned from the dataset in each iteration. For details, see the <a href="#">description of the SCAN command</a>. Iterative scanning can reduce the risks of slowing down Redis when a large number of keys are scanned at a time.</p> <p>For example, if there are 10 million keys in Redis and the number of keys to iterate is set to 1000, a full scan will be complete after 10,000 iterations.</p> <p>Value range: 10–1,000 Default value: 10 Unit: number</p>

**Step 7** After an expired key scan task is submitted, a task record is generated for each expired key scan. You can view the task ID, status, scan mode, start time, and end time.

 NOTE

The scan fails in the following scenarios:

- An exception occurred.
- There are too many keys, resulting in a timeout. Some keys have already been deleted before the timeout.

----End

## Automated Scan Performance and Suggestions

### Performance

- The **SCAN** command is executed at the data plane every 5 ms, that is, 200 times per second. If **Keys to Iterate** is set to **10**, **50**, **100**, or **1000**, 2000, 10,000, 20,000, or 200,000 keys are scanned per second.
- The larger the number of keys scanned per second, the higher the CPU usage.

### Reference test

A master/standby instance is scanned. There are 10 million keys that will not expire and 5 million keys that will expire. The expiration time is 1 to 10 seconds. A full scan is executed.

 NOTE

The following test results are for reference only. They may vary depending on the site environment and network fluctuation.

- Natural deletion: 10,000 expired keys are deleted per second. It takes 8 minutes to delete 5 million expired keys. The CPU usage is about 5%.
- **Keys to Iterate** set to **10**: The scanning takes 125 minutes (15 million/2000/60 seconds) and the CPU usage is about 8%.
- **Keys to Iterate** set to **50**: The scanning takes 25 minutes (15 million/10,000/60 seconds) and the CPU usage is about 10%.
- **Keys to Iterate** set to **100**: The scanning takes 12.5 minutes (15 million/20,000/60 seconds) and the CPU usage is about 20%.
- **Keys to Iterate** set to **1000**: The scanning takes 1.25 minutes (15 million/200,000/60 seconds) and the CPU usage is about 25%.

### Configuration suggestions

- You can configure the number of keys to be scanned and the scanning interval based on the total number of keys and the increase in the number of keys in the instance.
- In the reference test with 15 million keys and **Keys to Iterate** set to **10**, the scanning takes about 125 minutes. In this case, set the scan interval to more than 4 hours.
- If you want to accelerate the scanning, set **Keys to Iterate** to **100**. It takes about 12.5 minutes to complete the scanning. Therefore, set the scan interval to more than 30 minutes.
- The larger the number of keys to iterate, the faster the scanning, and the higher the CPU usage. There is a trade-off between time and CPU usage.

- If the number of expired keys does not increase rapidly, you can scan expired keys once a day.

 **NOTE**

Start scanning during off-peak hours. Set the interval to one day and the timeout to two days.


### 3.3.10 Managing IP Address Whitelist

DCS helps you control access to your DCS instances in the following ways, depending on the deployment mode:

- To control access to DCS Redis 3.0 instances, you can use security groups. Whitelists are not supported. For details about how to configure a security group, see .
- To control access to DCS Redis 4.0 and later instances, you can use whitelists. Security groups are not supported.

The following describes how to manage whitelists of a Redis 4.0 or later instance to allow access only from whitelisted IP addresses. If no whitelists are added for the instance or the whitelist function is disabled, all IP addresses that can communicate with the VPC can access the instance.

#### Creating a Whitelist Group

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of a DCS instance.
- Step 5** Choose **Instance Configuration > Whitelist** and then click **Create Whitelist Group**.
- Step 6** In the **Create Whitelist Group** dialogue box, specify **Group Name** and **IP Address/Range**.

**Table 3-7** Whitelist parameters

Parameter	Description	Example
Group Name	Whitelist group name of the instance.  A maximum of four whitelist groups can be created for each instance.	DCS-test

Parameter	Description	Example
IP Address/ Range	A maximum of 20 IP addresses or IP address ranges can be added to an instance. Separate multiple IP addresses or IP address ranges with commas.  Unsupported IP address and IP address range: 0.0.0.0 and 0.0.0.0/0.	10.10.10.1,10.10.10.10

**Step 7** Click **OK**.

A whitelist group is automatically enabled for the instance once created. Only whitelisted IP addresses can access the instance.

 **NOTE**

- In the whitelist group list, click **Modify** to modify the IP addresses or IP address ranges in a group, and click **Delete** to delete a whitelist group.
- After whitelist has been enabled, you can click **Disable Whitelist** above the whitelist group list to allow all IP addresses connected to the VPC to access the instance.

----End

### 3.3.11 Viewing Redis Run Logs

You can create run log files on the DCS console to collect run logs of DCS Redis instances within a specified period. After the logs are collected, you can download the log files to view the logs.

This function is supported by DCS Redis 4.0 instances and later.

#### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click the name of a DCS instance.

**Step 5** Click **Run Logs**.

**Step 6** Click **Create Log File**.

For a master/standby or cluster instance, logs will be collected from the specified shard and replica. If the instance is the single-node type, logs of the only node of the instance will be collected.

Select the collection period and click **OK**.

**Step 7** After the log file is successfully collected, click **Download** to download it.

----End

## 3.3.12 Diagnosing an Instance

### Scenario


If a fault or performance issue occurs, you can ask DCS to diagnose your instance to learn about the cause and impact of the issue and how to handle it.

### Restrictions

DCS Redis 3.0 instances do not support diagnosis.

### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner of the console and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click the name of a DCS Redis instance.

**Step 5** Choose **Analysis and Diagnosis > Instance Diagnosis**.

**Step 6** Specify the tested object and time range, and click **Start Diagnosis**.

- **Tested Object:** You can select a single node or all nodes. By default, all nodes are tested.
- **Range:** You can specify up to 10 minutes before a point in time in the last 7 days.

In the following figure, the data within 10 minutes before the specified time will be diagnosed.

#### NOTE

The system server time may be different from the current time. When the current time is selected, the actual diagnosis time range may be slightly shorter than the configured time range.

Instance diagnosis may fail during specification modification.

**Step 7** After the diagnosis is complete, you can view the result in the **Test History** list. If the result is abnormal, click **View Report** for details. You can click **Delete** to delete a record.

In the report, you can view the cause and impact of abnormal items and suggestions for handling them.

----End

## 3.4 Backing Up and Restoring DCS Instances

## 3.4.1 Overview

On the DCS console, you can back up and restore DCS instances.

### NOTE

By default, backup storage encryption is not enabled. Contact technical support to enable it.

## Importance of DCS Instance Backup

There is a small chance that inconsistent data could exist in a DCS instance owing to service system exceptions or problems in loading data from persistence files. In addition, some systems demand not only high reliability but also data security, data restoration, and even permanent data storage.

Currently, data in DCS instances can be backed up to OBS. If a DCS instance becomes faulty, data in the instance can be restored from backup so that service continuity is not affected.

## Impact on DCS Instances During Backup

**Backup tasks are run on standby cache nodes, without incurring any downtime.**

In the event of full synchronization of master and standby nodes or heavy instance load, it takes a few minutes to complete data synchronization. If instance backup starts before data synchronization is complete, the backup data will be slightly behind the data in the master cache node.

New data changes on the master node during an ongoing backup are not included in the backup.

## Backup Modes

DCS instances support the following backup modes:

- Automated backup

You can create a scheduled backup policy on the DCS console. Then, data in the chosen DCS instances will be automatically backed up at the scheduled time.

You can choose the days of the week on which scheduled backup will run. Backup data will be retained for a maximum of seven days. Backup data older than seven days will be automatically deleted.

The primary purpose of scheduled backups is to create complete data replicas of DCS instances so that the instance can be quickly restored if necessary.

- Manual backup

Backup requests can be issued manually. Data in the chosen DCS instances will be backed up to OBS.

Before performing high-risk operations, such as system maintenance or upgrade, back up DCS instance data.

When a DCS instance is in use, its backup data will not be automatically deleted. You can manually delete backup data as required. When you delete

the instance, its backup data is deleted along with the instance. If you need the backup data, download and save it in advance.

## Additional Information About Data Backup

- Instance type
  - Redis: Only master/standby, Proxy Cluster, and Redis Cluster instances can be backed up and restored, while single-node instances cannot. However, you can export data of a single-node instance to an RDB file using `redis-cli`. For details, see [Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?](#)
- Backup mechanisms

DCS for Redis 3.0 persists data with Redis AOF. DCS for Redis 4.0 and later persist data to RDB or AOF files in manual backup mode, and to RDB files in automatic backup mode.

Backup tasks run on standby cache nodes. DCS instance data is backed up by compressing and storing the data persistence files from the standby cache node to OBS.

DCS checks instance backup policies once an hour. If a backup policy is matched, DCS runs a backup task for the corresponding DCS instance.
- Impact on DCS instances during backup

Backup tasks run on standby cache nodes, without incurring any downtime.

In the event of full-data synchronization or heavy instance load, it takes a few minutes to complete data synchronization. If instance backup starts before data synchronization is complete, the backup data will be slightly behind the data in the master cache node.

During instance backup, the standby cache node stops persisting the latest changes to disk files. If new data is written to the master cache node during backup, the backup file will not contain the new data.
- Backup time

It is advisable to back up instance data during off-peak periods.
- Storage of backup files

Backup files are stored to OBS.
- Handling exceptions in scheduled backup

If a scheduled backup task is triggered while the DCS instance is restarting or being scaled up, the scheduled backup task will be run in the next cycle.

If backing up a DCS instance fails or the backup is postponed because another task is in progress, DCS will try to back up the instance in the next cycle. A maximum of three retries are allowed within a single day.
- Retention period of backup data

Scheduled backup files are retained for up to seven days. You can configure the retention period. At the end of the retention period, most backup files of the DCS instance will be automatically deleted, but at least one backup file will be retained.

Manual backup files are retained permanently and need to be manually deleted. When you delete the instance, its backup files will be deleted along with the instance. If you need to retain the backup data, download the backup files in advance.



## Data Restoration

- Data restoration process
  - a. You can initiate a data restoration request using the DCS console.
  - b. DCS obtains the backup file from OBS.
  - c. Read/write to the DCS instance is suspended.
  - d. The original data persistence file of the master cache node is replaced by the backup file.
  - e. The new data persistence file (that is, the backup file) is reloaded.
  - f. Data is restored, and the DCS instance starts to provide read/write service again.
- Impact on service systems

Restoration tasks run on master cache nodes. During restoration, data cannot be written into or read from instances.
- Handling data restoration exceptions

If a backup file is corrupted, DCS will try to fix the backup file while restoring instance data. If the backup file is successfully fixed, the restoration proceeds. If the backup file cannot be fixed, the master/standby DCS instance will be changed back to the state in which it was before data restoration.

### 3.4.2 Configuring an Automatic Backup Policy

On the DCS console, you can configure an automatic backup policy. The system then backs up data in your instances according to the backup policy.



By default, automatic backup is disabled. To enable it, perform the operations described in this section. Single-node instances do not support backup and restoration.

If automatic backup is not required, disable the automatic backup function in the backup policy.

#### Prerequisites

A master/standby or cluster DCS instance is in the **Running** state.

#### Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance to display more details about the DCS instance.
- Step 5** On the instance details page, click **Backups & Restorations**.
- Step 6** Slide  to the right to enable automatic backup. Backup policies will be displayed.

**Table 3-8** Parameters in a backup policy

Parameter	Description
Backup Schedule	Day of a week on which data in the chosen DCS instance is automatically backed up. You can select one or multiple days of a week.
Retention Period (days)	The number of days that automatically backed up data is retained. Backup data will be permanently deleted at the end of retention period and cannot be restored. Value range: 1-7.
Start Time	Time at which automatic backup starts. Value: the full hour between 00:00 to 23:00 The DCS checks backup policies once every hour. If the backup start time in a backup policy has arrived, data in the corresponding instance is backed up. <b>NOTE</b> Instance backup takes 5 to 30 minutes. The data added or modified during the backup process will not be backed up. To reduce the impact of backup on services, it is recommended that data should be backed up during off-peak periods. Only instances in the <b>Running</b> state can be backed up.

**Step 7** Click **OK**.

**Step 8** Automatic backup starts at the scheduled time. You can view backup records on the current page.

After the backup is complete, click **Download**, **Restore**, or **Delete** next to the backup record as required.

 **NOTE**

If the error "111400104 The DCS instance is recovering from an internal fault. Please try again." is displayed, contact O&M personnel.

----End

### 3.4.3 Manually Backing Up a DCS Instance


You need to manually back up data in DCS instances in a timely manner. This section describes how to manually back up data in master/standby instances using the DCS console.

By default, manually backed up data is permanently retained. If backup data is no longer in use, you can delete it manually.

#### Prerequisites

At least one master/standby DCS instance is in the **Running** state.

## Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance to display more details about the DCS instance.
- Step 5** On the instance details page, click **Backups & Restorations**.
- Step 6** Click **Create Backup**.
- Step 7** Select a backup file format.  
Only DCS Redis 4.0 and later instances support backup file format selection.
- Step 8** In the **Create Backup** dialog box, click **OK**.  
Information in the **Description** text box cannot exceed 128 bytes.  
After the backup is complete, click **Download**, **Restore**, or **Delete** next to the backup record as required.

### NOTE

Instance backup takes 10 to 15 minutes. The data added or modified during the backup process will not be backed up.

If the error "111400104 The DCS instance is recovering from an internal fault. Please try again." is displayed, do not back up the instance data. Provide the error information to O&M personnel.

----End


## 3.4.4 Restoring a DCS Instance

On the DCS console, you can restore backup data to a chosen DCS instance.

### Prerequisites

- At least one master/standby or cluster DCS instance is in the **Running** state.
- A backup task has been run to back up data in the instance to be restored and the status of the backup task is **Succeeded**.

## Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance to display more details about the DCS instance.
- Step 5** On the instance details page, click **Backups & Restorations**.

A list of historical backup tasks is then displayed.

**Step 6** Click **Restore** in the same row as the chosen backup task.

**Step 7** Click **OK** to start instance restoration.

Information in the **Description** text box cannot exceed 128 bytes.

The **Restoration History** tab page displays the result of the instance restoration task.

 **NOTE**

Instance restoration takes 1 to 30 minutes.

While being restored, DCS instances do not accept data operation requests from clients because existing data is being overwritten by the backup data.

If the error "111400104 The DCS instance is recovering from an internal fault. Please try again." is displayed, do not restore the instance data. Provide the error information to O&M personnel.

----End

### 3.4.5 Downloading a Backup File

Automatically backed up data can be retained for a maximum of 7 days. Manually backed up data is not free of charge and takes space in OBS. Due to these limitations, you are advised to download the RDB and AOF backup files and permanently save them on the local host.


This function is supported only by master/standby and cluster instances, and not by single-node instances. To export the data of a single-node instance to an RDB file, you can use redis-cli. For details, see [How Do I Export DCS Redis Instance Data?](#)

#### Prerequisites

The instance has been backed up and the backup is still valid.

#### Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

Filter DCS instances to find the desired one.

**Step 4** Click the name of the DCS instance to display more details about the DCS instance.

**Step 5** On the instance details page, click **Backups & Restorations**.

A list of historical backup tasks is then displayed.

**Step 6** Select the historical backup data to be downloaded, and click **Download**.

**Step 7** In the displayed, **Download Backup File** dialog box, select either of the following two download methods.

Download methods:

- By URL
  - a. Set the URL validity period and click **Query**.
  - b. Download the backup file by using the URL list.

 **NOTE**

If you choose to copy URLs, use quotation marks to quote the URLs when running the **wget** command in Linux. For example:

```
wget 'https://obsEndpoint.com:443/redisdemo.rdb?  
parm01=value01&parm02=value02'
```

This is because the URL contains the special character and (&), which will confuse the **wget** command. Quoting the URL facilitates URL identification.

- By OBS  
Perform the procedure as prompted.

----End

## 3.5 Migrating Data with DCS

### 3.5.1 Introduction to Migration with DCS

#### Migration Modes

DCS for Redis supports online migration (in full or incrementally) and backup migration (by importing backup files).

- Backup migration is suitable when the source and target Redis instances are not connected, and the source Redis instance does not support the **SYNC** and **PSYNC** commands. The data source can be an OBS bucket or a Redis instance.
  - Importing data from an OBS bucket: Download the source Redis data and then upload it to an OBS bucket in the same region as the target DCS Redis instance. DCS will read the backup data from the OBS bucket and migrate the data into the target instance.  
**This migration mode can be used for migrating data from other Redis vendors or self-hosted Redis to DCS for Redis.**
  - Importing data from a Redis instance: Back up the source Redis data and then migrate the backup data to DCS for Redis.
- Migrating data online: If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported in the source instance, data can be migrated online in full or incrementally from the source to the target.

The following table describes data migration modes supported by DCS.

 **NOTE**

Data can be migrated only from DCS Redis instances or self-hosted Redis. After data migration, change the instance connection address to the target instance address.

Data migration is not supported if the DCS instance is created by another service, such as ROMA Connect, or by calling an API.

**Table 3-9** DCS data migration modes

Migration Mode	Source	Target: DCS		
		Single-node or master/standby	Proxy Cluster	Redis Cluster
Importing backup files	AOF/RDB file	√	√	√
Migrating data online	DCS for Redis: Single-node or master/standby	√	√	√
	DCS for Redis: Proxy Cluster <b>NOTE</b> Proxy Cluster DCS Redis 3.0 instances cannot be used as the source, while Proxy Cluster DCS Redis 4.0 or 5.0 instances can.	√	√	√
	DCS for Redis: Redis Cluster	√	√	√
	Self-hosted Redis: single-node or master/standby	√	√	√
	Self-hosted Redis: proxy-based cluster	√	√	√
	Self-hosted Redis: Redis Cluster	√	√	√
	Other Redis: single-node or master/standby	√	√	√
	Other Redis: proxy-based cluster	√	√	√

	Other Redis: Redis Cluster	√	√	√
	<p><b>NOTE</b> You can migrate data online in full or incrementally from <b>other cloud Redis</b> to <b>DCS for Redis</b> if they are connected and the <b>SYNC</b> and <b>PSYNC</b> commands can be run on the source Redis. However, some instances provided by other cloud vendors may fail to be migrated online. In this case, migrate data through backup import or use other migration schemes.</p>			

 **NOTE**

- **DCS for Redis** refers to Redis instances provided by DCS
- **Self-hosted Redis** refers to self-hosted Redis on the cloud, from other cloud vendors, or in on-premises data centers.
- **Other Redis** refers to Redis services provided by other cloud vendors.
- √: Supported. ×: Not supported.

## 3.5.2 Importing Backup Files

### 3.5.2.1 Importing Backup Files from an OBS Bucket

#### Scenario

Use the DCS console to migrate backup data to DCS for Redis.

Simply download the source Redis backup data and then upload the data to an OBS bucket in the same region as the target DCS Redis instance. After you have created a migration task on the DCS console, DCS will read data from the OBS bucket and data will be migrated to the target instance.

.aof, .rbb, .zip, and .tar.gz files can be uploaded to OBS buckets. You can directly upload .aof and .rdb files or compress them into .zip or .tar.gz files before uploading.

#### Prerequisites

- The OBS bucket must be in the same region as the target DCS Redis instance.
- The data files to be uploaded must be in .aof, .rdb, .zip, or .tar.gz formats. .zip files must contain .aof or .rdb files.
- To migrate data from a single-node or master/standby Redis instance of other cloud vendors, create a backup task and download the backup file.
- To migrate data from a cluster Redis instance of other cloud vendors, download all backup files and upload all of them to the OBS bucket. Each backup file contains data for a shard of the instance.

 **NOTE**

1. Importing backups generated by a later-version Redis instance to an earlier one may fail.
2. Before importing backup files, ensure that resource-intensive commands (such as FLUSHALL, KEYS, and HGETALL) have been disabled on the target Redis.
3. The Redis memory usage of a single .rdb backup file must be less than 10 GB.
4. If the backup file contains multi-DB data, its database count cannot exceed what is supported by the target Redis.
5. Multi-DB backup files generated from Proxy Cluster instances cannot be imported.

## Preparing the Target DCS Redis Instance

- If a DCS Redis instance is not available, create one first. For details, see [Creating a DCS Redis Instance](#).
- If a DCS Redis instance is available, you do not need to create a new one. However, you can clear the instance data before the migration.
  - If the target instance is Redis 4.0 and later, clear the data by referring to [Clearing DCS Instance Data](#).
  - If the target instance is a DCS Redis 3.0 instance, run the **FLUSHALL** command to clear data.
  - If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.
- Redis is backward compatible. The target instance version must be the same as or later than the source instance version.

## Creating an OBS Bucket and Uploading Backup Files

### Step 1 Create an OBS bucket.

1. Log in to the OBS Console and click **Create Bucket**.
2. Select a region.  
The OBS bucket must be in the same region as the target DCS Redis instance.
3. Specify **Bucket Name**.  
The bucket name must meet the naming rules specified on the console.
4. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
5. Click **Create Now**.

### Step 2 Upload the backup data files to the OBS bucket by using OBS Browser+.

If the backup file to be uploaded does not exceed 5 GB, upload the file using the OBS console by referring to step [Step 3](#).

If the backup file to be uploaded is larger than 5 GB, perform the following steps to upload the file using OBS Browser+.

1. Download OBS Browser+.  
For details, see section "Downloading OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+) > "Getting Started"*.



2. Install OBS Browser+.  
For details, see section "Installing OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)* > "Getting Started".
3. Log in to OBS Browser+.  
For details, see section "Logging In to OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)* > "Getting Started".
4. Create a bucket.
5. Upload backup data.

**Step 3** On the OBS console, upload the backup data files to the OBS bucket.


Perform the following steps if the backup file size does not exceed 5 GB:

1. In the bucket list, click the name of the created bucket.
2. In the navigation pane, choose **Objects**.
3. On the **Objects** tab page, click **Upload Object**.
4. Upload the objects.  
To upload objects, drag files or folders to the **Upload Object** area or click **add file**. A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.
5. Click **Upload**.

----End

## Creating a Migration Task

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Data Migration**.

**Step 4** Click **Create Backup Import Task**.

**Step 5** Specify **Task Name** and **Description**.

**Step 6** Select **OBS Bucket** as the data source and then select the OBS bucket to which you have uploaded backup files.

### NOTE

You can upload files in the .aof, .rdb, .zip, or .tar.gz format.

**Step 7** In the **Backup Files** area, click **Add Backup** and select the backup files to be migrated.

**Step 8** Select the target DCS Redis instance prepared in [Preparing the Target DCS Redis Instance](#).

**Step 9** Enter the password of the target instance. Click **Test Connection** to verify the password. If the instance is not password-protected, click **Test Connection** directly.

**Step 10** Click **Next**.

**Step 11** Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

 **NOTE**

If the error "DCS.4104 The DCS instance is recovering from an internal fault. Please try again." is displayed, contact O&M personnel.

----End

### 3.5.2.2 Importing Backup Files from Redis

#### Scenario

Use the DCS console to migrate Redis data from self-hosted Redis to DCS for Redis.

Simply back up your Redis data, create a migration task on the DCS console, and then import the backup to a DCS Redis instance.

#### Prerequisites

A master/standby or cluster DCS Redis instance has been created as the target for the migration. The source instance has data and has been backed up.

 **NOTE**

1. Importing backups generated by a later-version Redis instance to an earlier one may fail.
2. Before importing backup files, ensure that resource-intensive commands (such as FLUSHALL, KEYS, and HGETALL) have been disabled on the target Redis.
3. The Redis memory usage of a single .rdb backup file must be less than 10 GB.
4. If the backup file contains multi-DB data, its database count cannot exceed what is supported by the target Redis.
5. Multi-DB backup files generated from Proxy Cluster instances cannot be imported.


#### Step 1: Obtain the Source Instance Name and Password

Obtain the name of the source Redis instance.

#### Step 2: Prepare the Target DCS Redis Instance

- If a DCS Redis instance is not available, create one first. For details, see [Creating a DCS Redis Instance](#).
- If a DCS Redis instance is available, you do not need to create a new one. However, you can clear the instance data before the migration.
  - If the target instance is Redis 4.0 and later, clear the data by referring to [Clearing DCS Instance Data](#).
  - If the target instance is a DCS Redis 3.0 instance, run the **FLUSHALL** command to clear data.
  - If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

## Step 3: Create a Migration Task

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Data Migration**.
- Step 4** Click **Create Backup Import Task**.
- Step 5** Enter the task name and description.
- Step 6** Set **Data Source** to **Redis**.
- Step 7** For source Redis, select the instance prepared in [Step 1: Obtain the Source Instance Name and Password](#).
- Step 8** Select the backup task whose data is to be migrated.
- Step 9** For **Target Instance**, select the DCS Redis prepared in [Step 2: Prepare the Target DCS Redis Instance](#).
- Step 10** Enter the password of the target instance. Click **Test Connection** to verify the password. If the instance is not password-protected, click **Test Connection** directly.
- Step 11** Click **Next**.
- Step 12** Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

## 3.5.3 Migrating Data Online

### Scenario

If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported in the source instance, data can be migrated online in full or incrementally from the source to the target.

---

#### CAUTION

- If the **SYNC** and **PSYNC** commands are disabled on the source Redis instance, enable them before performing online migration. Otherwise, the migration fails. If you use a DCS Redis instance for online migration, the **SYNC** command is automatically enabled.
- During online migration, you are advised to set **repl-timeout** on the source instance to 300s and **client-output-buffer-limit** to 20% of the maximum memory of the instance.

---

#### NOTE

During online migration, results of the **FLUSHDB** and **FLUSHALL** commands executed on the source will not be synchronized to the target.

## Impacts on Services

During online migration, data is essentially synchronized in full to a new replica. Therefore, perform online migration during low-demand hours.

## Prerequisites

- Before migrating data, read through [Introduction to Migration with DCS](#) to learn about the DCS data migration function and select an appropriate target instance.
- To migrate data from a single-node or master/standby instance to a cluster instance, check if any data exists in DBs other than DB0 in the source instance. If yes, move the data to DB0 by using the open-source tool Rump. Otherwise, the migration will fail because a cluster instance has only one DB. For details about the migration operations, see [Online Migration with Rump](#).

## Step 1: Obtain Information About the Source Redis Instance

- If the source is a cloud Redis instance, obtain its name.
- If the source is self-hosted Redis, obtain its IP address or domain name and port number.

## Step 2: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see [Creating a DCS Redis Instance](#).
- If a target instance is available, you do not need to create a new one. However, you must clear the instance data before the migration. For details, see [Clearing DCS Instance Data](#).

If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

## Step 3: Check the Network

### NOTE


- If the source or target of online migration is **Redis in the cloud**, the selected Redis instance must be in the same VPC as the migration task. Otherwise, the migration task may fail to connect to the cloud Redis instance.
- In special scenarios, if you have enabled cross-VPC access between the migration task and the cloud Redis instance, the cloud Redis instance and the migration task can be in different VPCs.

[Table 3-10](#) lists the requirements on the network between the online migration task, source Redis, and target Redis.

**Table 3-10** Requirements on the network between the online migration task, source Redis, and target Redis

Source Redis Type	Target Redis Type	Network Requirement on Online Migration
Redis in the cloud	Redis in the cloud	When creating an online migration task, ensure that the online migration task is in the same VPC as the source and target Redis. If they are not in the same VPC, enable cross-network access between the migration task and the source and target Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .
Redis in the cloud	Self-hosted Redis	When creating an online migration task, ensure that the migration task and the source Redis are in the same VPC. Then, enable cross-network access between the migration task and the target Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .
Self-hosted Redis	Redis in the cloud	When creating an online migration task, ensure that the migration task and the target Redis are in the same VPC. Then, enable cross-network access between the migration task and the source Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .
Self-hosted Redis	Self-hosted Redis	After creating an online migration task, enable cross-network access between the migration task and the source and target Redis, respectively. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .

## Step 4: Create a Migration Task

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Data Migration**. The migration task list is displayed.
- Step 4** Click **Create Online Migration Task**.
- Step 5** Enter the task name and description.

**Step 6** Configure the VPC, subnet, and security group for the migration task.

The VPC, subnet, and security group facilitate the migration. Ensure that the migration resources can access the source and target Redis instances.

**NOTICE**

- The migration task uses a tenant IP address (**Migration ECS** displayed on the **Basic Information** page of the task.) If a whitelist is configured for the source or target instance, add the migration IP address to the whitelist or disable the whitelist.
- To allow the VM used by the migration task to access the source and target instances, set an outbound rule for the task's security group to allow traffic through the IP addresses and ports of the source and target instances. By default, all outbound traffic is allowed.

**Step 7** Click **Next**.

**Step 8** Click **Submit**.

----End

## Configuring the Online Migration Task

**Step 1** On the **Online Migration** tab page, click **Configure** in the row containing the online migration task you just created.

**Step 2** Specify **Migration Type**.

Supported migration types are **Full** and **Full + incremental**, which are described in [Table 3-11](#).

**Table 3-11** Migration type description

Migration Type	Description
Full	Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source instance data updated during the migration will not be migrated to the target instance.
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and target instances.  <b>Once incremental migration starts, it remains Migrating</b> until you click <b>Stop</b> in the <b>Operation</b> column. After the migration is stopped, data in the source instance will not be lost, but data will not be written to the target instance. When the transmission network is stable, the delay of incremental migration is within seconds. The actual delay depends on the transmission quality of the network link.

**Step 3** Configure source Redis and target Redis.

1. The Redis type can be **Redis in the cloud** or **Self-hosted Redis** as required.
  - **Redis in the cloud:** a DCS Redis instance that is in the same VPC as the migration task
  - **Self-hosted Redis:** self-hosted Redis in another cloud, or in on-premises data centers. If you select this option, enter Redis addresses.

 **NOTE**

If the source and target Redis instances are connected but are in different regions of DCS, you can only select **Self-hosted Redis** for **Target Redis Type** and enter the instance addresses, regardless of whether the target Redis instance is self-hosted or in the cloud.

2. If the instance is password-protected, you can click **Test Connection** to check whether the instance password is correct and whether the network is connected. If the instance is not password-protected, click **Test Connection** directly.

**Step 4** Click **Next**.

**Step 5** Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

 **NOTE**

- If the migration type is full+incremental, the migration task status will remain **Migrating** until you click **Stop**.
- If the error "DCS.4104 The DCS instance is recovering from an internal fault. Please try again." is displayed, do not migrate data. Contact O&M personnel to handle the exception.
- After data migration, duplicate keys will be overwritten.

----End

## Verifying the Migration

After the migration is complete, use redis-cli to connect the source and target Redis instances to check data integrity.

1. Connect to the source Redis and the target Redis.
2. Run the **info keyspace** command to check the values of **keys** and **expires**.

```
192.168.1.217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.168.1.217:6379>
```

3. Calculate the differences between the values of **keys** and **expires** of the source Redis and the target Redis. If the differences are the same, the data is complete and the migration is successful.

During full migration, source Redis data updated during the migration will not be migrated to the target instance.

## 3.6 Managing Passwords

### 3.6.1 DCS Instance Passwords

Passwords can be configured to control access to your DCS instances, ensuring the security of your data.

#### NOTE

After 5 consecutive incorrect password attempts, the account for accessing the chosen DCS instance will be locked for 5 minutes. Passwords cannot be changed during the lockout period.

The password must meet the following requirements:

- Cannot be left blank.
- Cannot be the same as the old password.
- Can contain 8 to 64 characters.
- Must contain at least three of the following character types:
  - Lowercase letters
  - Uppercase letters
  - Digits
  - Special characters (``~!@#$%^&*()-_+=+\\{|},<.>/?`)

### Using Passwords Securely

1. Hide the password when using redis-cli.

If the `-a <password>` option is used in redis-cli in Linux, the password is prone to leakage because it is logged and kept in the history. You are advised not to use `-a <password>` when running commands in redis-cli. After connecting to Redis, run the `auth` command to complete authentication as shown in the following example:

```
$ redis-cli -h 192.168.0.148 -p 6379
redis 192.168.0.148:6379>auth {yourPassword}
OK
redis 192.168.0.148:6379>
```

2. Use interactive password authentication or switch between users with different permissions.

If the script involves DCS instance access, use interactive password authentication. To enable automatic script execution, manage the script as another user and authorize execution using `sudo`.

3. Use an encryption module in your application to encrypt the password.

### 3.6.2 Changing Instance Passwords

On the DCS console, you can change the password required for accessing your DCS instance.



 **NOTE**


- You cannot change the password of a DCS instance in password-free mode.
- The DCS instance for which you want to change the password is in the **Running** state.
- The new password takes effect immediately on the server without requiring a restart. The client must reconnect to the server using the new password after a pconnect connection is closed. (The old password can still be used before disconnection.)

## Prerequisites

At least one DCS instance has been created.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Choose **More > Change Password** in the same row as the chosen instance.

**Step 5** In the displayed dialog box, set **Old Password**, **New Password**, and **Confirm Password**.

 **NOTE**

After 5 consecutive incorrect password attempts, the account for accessing the chosen DCS instance will be locked for 5 minutes. Passwords cannot be changed during the lockout period.

The password must meet the following requirements:

- Cannot be left blank.
- The new password cannot be the same as the old password.
- Can contain 8 to 64 characters.
- Must contain at least three of the following character types:
  - Lowercase letters
  - Uppercase letters
  - Digits
  - Special characters (`~!@#$%^&*()-_+=+\\{|},<.>/?`)

**Step 6** In the **Change Password** dialog box, click **OK** to confirm the password change.

----End

## 3.6.3 Resetting Instance Passwords

On the DCS console, you can configure a new password if you forget your instance password.

 **NOTE**


- For a DCS Redis instance, you can change it from password mode to password-free mode or from password-free mode to password mode by resetting its password. For details, see [Changing Password Settings for DCS Redis Instances](#).
- The DCS instance for which you want to reset the password is in the **Running** state.
- The new password takes effect immediately on the server without requiring a restart. The client must reconnect to the server using the new password after a pconnect connection is closed. (The old password can still be used before disconnection.)

## Prerequisites

At least one DCS instance has been created.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Choose **More > Reset Password** in the row containing the chosen instance.

**Step 5** In the **Reset Password** dialog box, enter a new password and confirm the password.

 **NOTE**

The password must meet the following requirements:

- Cannot be left blank.
- Can contain 8 to 64 characters.
- Contain at least three of the following character types:
  - Lowercase letters
  - Uppercase letters
  - Digits
  - Special characters (`~!@#$%^&*()-_+=\|{};<.>/?`)

**Step 6** Click **OK**.

 **NOTE**

The system will display a success message only after the password is successfully reset on all nodes. If the reset fails, the instance will restart and the password of the cache instance will be restored.

----End

## 3.6.4 Changing Password Settings for DCS Redis Instances

### Scenario

DCS Redis instances can be accessed with or without passwords. After an instance is created, you can change its password setting in the following scenarios:

- To access a DCS Redis instance in password-free mode, you can enable password-free access to clear the existing password of the instance.

 **NOTE**

- To change the password setting, the DCS Redis instance must be in the **Running** state.
- Password-free access may compromise security. You can set a password by using the password reset function.

## Procedure

**Step 1** Log in to the DCS console.

**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** To change the password setting for a DCS Redis instance, choose **Operation > More > Reset Password** in the same row as the chosen instance.

**Step 5** In the **Reset Password** dialogue box, perform either of the following operations as required:

- From password-protected to password-free:  
Switch the toggle for **Password-Free Access** and click **OK**.
- From password-free to password-protected:  
Enter a password, confirm the password, and click **OK**.

----End

## 3.7 Monitoring

### 3.7.1 DCS Metrics

#### Introduction

This section describes DCS metrics reported to Cloud Eye as well as their namespaces and dimensions. You can use the Cloud Eye console or call APIs to query the DCS metrics and alarms.

**Table 3-12** Monitoring dimensions for different instance types

Instance Type	Instance Monitoring	Redis Server Monitoring	Proxy Monitoring
Single-node	Supported The monitoring on the instance dimension is conducted on the Redis Server.	N/A	N/A

Instance Type	Instance Monitoring	Redis Server Monitoring	Proxy Monitoring
Master/standby	Supported The master node is monitored.	Supported The master and standby nodes are monitored.	N/A
Proxy Cluster	Supported The monitoring data is the aggregated master node data.	Supported Each shard is monitored.	Supported Each proxy is monitored.
Redis Cluster	Supported The monitoring data is the aggregated master node data.	Supported Each shard is monitored.	N/A

## Namespace

SYS.DCS

## DCS Redis 3.0 Instance Metrics

### NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

**Table 3-13** DCS Redis 3.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Single-node, master/standby, or cluster DCS Redis instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Single-node, master/standby, or cluster DCS Redis instance	1 minute
net_in_throughput	Network Input Throughput	Inbound throughput per second on a port Unit: byte/s	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
net_out_throughput	Network Output Throughput	Outbound throughput per second on a port Unit: byte/s	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
node_status	Instance Node Status	Status of instance nodes. If the status is normal, the value is <b>0</b> . If the status is abnormal, the value is <b>1</b> .	-	Single-node, master/standby, or cluster DCS Redis instance	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
memory_fragment_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between <b>used_memory_rss/used_memory</b> .	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: kbit/s	$\geq 0$ kbits/s	Single-node, master/standby, or cluster DCS Redis instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: kbit/s	$\geq 0$ kbits/s	Single-node, master/standby, or cluster DCS Redis instance	1 minute
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
keyspace_hits	Keyspace Hits	Number of successful lookups of keys in the main dictionary during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute



Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	$\geq 0$	Single-node, master/standby, or cluster DCS Redis instance	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: $\text{keyspace\_hits} / (\text{keyspace\_hits} + \text{keyspace\_misses})$ Unit: %	0–100%	Single-node, master/standby, or cluster DCS Redis instance	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	$\geq 0$ ms	Single-node, master/standby, or cluster DCS Redis instance	1 minute
auth_errors	Authentication Failures	Number of failed authentications	$\geq 0$	Single-node or master/standby DCS Redis instance	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance <b>NOTE</b> Slow queries caused by the <b>MIGRATE</b> , <b>SLAVEOF</b> , <b>CONFIG</b> , <b>BGSAVE</b> , and <b>BGREWRITEAOF</b> commands are not counted.	<ul style="list-style-type: none"> <li>• <b>1</b>: yes</li> <li>• <b>0</b>: no</li> </ul>	Single-node or master/standby DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
keys	Keys	Number of keys in Redis	≥ 0	Single-node or master/standby DCS Redis instance	1 minute

## DCS Redis 4.0/5.0/6.0 Instance Metrics

 NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

**Table 3-14** DCS Redis 4.0/5.0/6.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Single-node or master/standby DCS Redis instance	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
total_connections_received	New Connections	Number of connections received during the monitoring period	$\geq 0$	DCS Redis instance	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance <b>NOTE</b> Slow queries caused by the <b>MIGRATE</b> , <b>SLAVEOF</b> , <b>CONFIG</b> , <b>BGSAVE</b> , and <b>BGREWRITEAOF</b> commands are not counted.	<ul style="list-style-type: none"> <li>• <b>1</b>: yes</li> <li>• <b>0</b>: no</li> </ul>	DCS Redis instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	DCS Redis instance	1 minute
expires	Keys With an Expiration	Number of keys with an expiration in Redis	$\geq 0$	DCS Redis instance	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: $\text{keyspace\_hits} / (\text{keyspace\_hits} + \text{keyspace\_misses})$ Unit: %	0–100%	DCS Redis instance	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: KB, MB, or byte (configurable on the console)	$\geq 0$	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: KB, MB, or byte (configurable on the console)	$\geq 0$	DCS Redis instance	1 minute
used_memory_dataset_perc	Used Memory Dataset Ratio	Percentage of dataset memory that server has used Unit: %	0–100%	DCS Redis instance	1 minute
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: KB, MB, or byte (configurable on the console)	$\geq 0$	DCS Redis instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	$\geq 0$	DCS Redis instance	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	$\geq 0$	DCS Redis instance	1 minute
keys	Keys	Number of keys in Redis	$\geq 0$	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period	$\geq 0$	DCS Redis instance	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the maximum bandwidth limit used (the average value of the sum of input and output flows) Unit: %	$\geq 0$	DCS Redis instance	1 minute
command_max_rt	Maximum Latency	Maximum delay from when the node receives commands to when it responds Unit: $\mu\text{s}$	$\geq 0$	Single-node DCS Redis instance	1 minute
command_avg_rt	Average Latency	Average delay from when the node receives commands to when it responds Unit: $\mu\text{s}$	$\geq 0$	Single-node DCS Redis instance	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations	$\geq 0$	DCS Redis instance	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	$\geq 0$	DCS Redis instance	1 minute
del	DEL	Number of <b>DEL</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	DCS Redis instance	1 minute
expire	EXPIRE	Number of <b>EXPIRE</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	DCS Redis instance	1 minute
get	GET	Number of <b>GET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hdel	HDEL	Number of <b>HDEL</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hget	HGET	Number of <b>HGET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hmget	HMGET	Number of <b>HMGET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hmset	HMSET	Number of <b>HMSET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
hset	HSET	Number of <b>HSET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	DCS Redis instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	DCS Redis instance	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Ratio between Used Memory RSS and Used Memory	≥ 0	DCS Redis instance	1 minute
mget	MGET	Number of <b>MGET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
mset	MSET	Number of <b>MSET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	DCS Redis instance	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	DCS Redis instance	1 minute
set	SET	Number of <b>SET</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
sadd	SADD	Number of <b>SADD</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
smembers	SMEMBERS	Number of <b>SMEMBERS</b> commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	DCS Redis instance	1 minute
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute



Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_dataset_percent	Used Memory Dataset Ratio	Percentage of dataset memory that server has used Unit: %	0–100%	DCS Redis instance	1 minute

## Redis Server Metrics of DCS Redis Instances

 NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

**Table 3-15** Redis Server metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Redis Server of a cluster instance  Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	$\geq 0$	Redis Server of a master/standby or cluster DCS Redis 4.0 or later instance	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	$\geq 0$	Redis Server of a master/standby or cluster DCS Redis 4.0 or later instance	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
used_memory_rss	Used Memory RSS	RSS memory that the Redis server has used, which including all stack and heap memory but not swapped-out memory Unit: byte	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between <b>used_memory_rss/used_memory</b> .	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	$\geq 0$ KB/s	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	$\geq 0$ KB/s	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	$\geq 0$	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: $\text{keyspace\_hits} / (\text{keyspace\_hits} + \text{keyspace\_misses})$ Unit: %	0–100%	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the node <b>NOTE</b> Slow queries caused by the <b>MIGRATE</b> , <b>SLAVEOF</b> , <b>CONFIG</b> , <b>BGSAVE</b> , and <b>BGREWRITEAOF</b> commands are not counted.	<ul style="list-style-type: none"> <li>• <b>1</b>: yes</li> <li>• <b>0</b>: no</li> </ul>	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
keys	Keys	Number of keys in Redis	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
sadd	SADD	Number of <b>SADD</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
smembers	SMEMBERS	Number of <b>SMEMBERS</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
ms_repl_offset	Replication Gap	Data synchronization gap between the master and the replica	-	Replica of a cluster DCS Redis 4.0 or 5.0 instance	1 minute
del	DEL	Number of <b>DEL</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute



Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
expire	EXPIRE	Number of <b>EXPIRE</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
get	GET	Number of <b>GET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
hdel	HDEL	Number of <b>HDEL</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
hget	HGET	Number of <b>HGET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
hmget	HMGET	Number of <b>HMGET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
hmset	HMSET	Number of <b>HMSET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
hset	HSET	Number of <b>HSET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
mget	MGET	Number of <b>MGET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
mset	MSET	Number of <b>MSET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
set	SET	Number of <b>SET</b> commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period Unit: count	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0 or later instance	1 minute

## Proxy Metrics

 NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

**Table 3-16** Proxy metrics of Proxy Cluster DCS Redis 3.0 instances

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0-100%	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0-100%	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
p_connected_clients	Connected Clients	Number of connected clients	≥ 0	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
max_rxpck_per_sec	Max. NIC Data Packet Receive Rate	Maximum number of data packets received by the proxy NIC per second during the monitoring period Unit: packages/second	0-10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
max_txpck_per_sec	Max. NIC Data Packet Transmit Rate	Maximum number of data packets transmitted by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
max_rxB_per_sec	Maximum Inbound Bandwidth	Largest volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
max_txkB_per_sec	Maximum Outbound Bandwidth	Largest volume of data transmitted by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
avg_rxpck_per_sec	Average NIC Data Packet Receive Rate	Average number of data packets received by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
avg_txpck_per_sec	Average NIC Data Packet Transmit Rate	Average number of data packets transmitted by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
avg_rxB_per_sec	Average Inbound Bandwidth	Average volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
avg_txkB_per_sec	Average Outbound Bandwidth	Average volume of data transmitted by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute

**Table 3-17** Proxy metrics of Proxy Cluster DCS Redis 4.0 or 5.0 instances

Metric ID	Metric	Metric Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
node_status	Instance Node Status	Indication of whether the proxy is normal.	<ul style="list-style-type: none"> <li>• <b>0:</b> Normal</li> <li>• <b>1:</b> Abnormal</li> </ul>	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0-100%	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute

Metric ID	Metric	Metric Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0-100%	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
connected_clients	Connected Clients	Number of connected clients	≥ 0	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute

Metric ID	Metric	Metric Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
connections_usage	Connection Usage	Percentage of the current number of connections to the maximum allowed number of connections Unit: %	0-100%	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
command_max_rt	Maximum Latency	Maximum delay from when the node receives commands to when it responds Unit: us	>=0us	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute
command_avg_rt	Average Latency	Average delay from when the node receives commands to when it responds Unit: us	>=0us	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance	1 minute

## Dimensions

Key	Value
dc_instance_id	DCS Redis instance
dc_cluster_redis_node	Redis Server
dc_cluster_proxy_node	Proxy in a Proxy Cluster DCS Redis 3.0 instance
dc_cluster_proxy2_node	Proxy in a Proxy Cluster DCS Redis 4.0 or 5.0 instance


## 3.7.2 Viewing DCS Monitoring Metrics

You can view DCS instance metrics on the **Performance Monitoring** page.

### Procedure

**Step 1** Log in to the DCS console.



**Step 2** Click  in the upper left corner and select a region and a project.

**Step 3** In the navigation pane, choose **Cache Manager**.

**Step 4** Click the desired instance.

**Step 5** Choose **Performance Monitoring**. All monitoring metrics of the instance are displayed.

 **NOTE**

You can also click **View Metric** in the **Operation** column on the **Cache Manager** page. You will be redirected to the Cloud Eye console. The metrics displayed on the Cloud Eye console are the same as those displayed on the **Performance Monitoring** page of the DCS console.

----End

# 4 Best Practices

---

## 4.1 Service Application

### 4.1.1 Serializing Access to Frequently Accessed Resources

#### Overview

##### Application Scenario

In monolithic deployment, you can use Java concurrency APIs such as **ReentrantLock** or **synchronized** to implement mutual exclusion locks. This native lock mechanism provided by Java ensures that multiple threads within a Java VM process are executed concurrently and sequentially.

However, this mechanism may fail in multi-node deployment because a node's lock only takes effect on threads in the Java VM where the node runs. For example, the concurrency level in Internet seckill requires multiple nodes to run at the same time. Assume that requests of two users arrive simultaneously on two nodes. Although the requests can be processed simultaneously on the respective nodes, an inventory oversold problem may still occur because the nodes use different locks.

##### Solution

To serialize access to resources, ensure that all nodes use the same lock. This requires a distributed lock.

The idea of a distributed lock is to provide a globally unique "thing" for different systems to allocate locks. When a system needs a lock, it asks the "thing" for a lock. In this way, different systems can obtain the same lock.

Currently, a distributed lock can be implemented using cache databases, disk databases, or ZooKeeper.

Implementing distributed locks using DCS Redis instances has the following advantages:

- Simple operation: Locks can be acquired and released by using simple commands such as **SET**, **GET**, and **DEL**.

- High performance: Cache databases deliver higher read/write performance than disk databases and ZooKeeper.
- High reliability: DCS supports both master/standby and cluster instances, preventing single points of failure.

Implementing locks on distributed applications can avoid inventory oversold problems and nonsequential access. The following describes how to implement locks on distributed applications with Redis.

## Prerequisites

- A DCS instance has been created, and is in the **Running** state.
- The network between the client server and the DCS instance is connected:
  - When the client and the DCS Redis instance are in the same VPC:  
By default, networks in a VPC can communicate with each other.
  - When the client and the DCS Redis instance are in different VPCs in the same region:  
If the client and DCS Redis instance are not in the same VPC, connect them by establishing a VPC peering connection. For details, see "Does DCS Support Cross-VPC Access?" in *Distributed Cache Service User Guide* > FAQs.
  - To access a Redis instance of another region on a client  
If the client server and the Redis instance are not in the same region, connect the network using Direct Connect. For details, see *Direct Connect User Guide*.
- You have installed **JDK1.8** (or later) and a development tool (**Eclipse** is used as an example) on the client server, and downloaded the **Jedis client**.  
The development tools and clients mentioned in this document are for example only.

## Procedure

- Step 1** Run Eclipse on the server and create a Java project. Then, create a distributed lock implementation class **DistributedLock.java** and a test class **CaseTest.java** for the example code, and reference the Jedis client as a library to the project.

Sample code of **DistributedLock.java**:

```
package dcsDemo01;

import java.util.UUID;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.params.SetParams;

public class DistributedLock {
    // Address and port for connecting to the Redis instance. Replace them with the actual values.
    private final String host = "192.168.0.220";
    private final int port = 6379;

    private static final String SUCCESS = "OK";

    public DistributedLock(){}

    /*
     * @param lockName    Lock name
    */
}
```

```

    * @param timeout    Timeout for acquiring locks
    * @param lockTimeout Validity period of locks
    * @return          Lock ID
    */
public String getLockWithTimeout(String lockName, long timeout, long lockTimeout) {
    String ret = null;
    Jedis jedisClient = new Jedis(host, port);

    try {
        // Password for connecting to the Redis instance. Replace it with the actual value.
        String authMsg = jedisClient.auth("passwd");
        if (!SUCCESS.equals(authMsg)) {
            System.out.println("AUTH FAILED: " + authMsg);
        }

        String identifier = UUID.randomUUID().toString();
        String lockKey = "DLock:" + lockName;
        long end = System.currentTimeMillis() + timeout;

        SetParams setParams = new SetParams();
        setParams.nx().px(lockTimeout);

        while(System.currentTimeMillis() < end) {
            String result = jedisClient.set(lockKey, identifier, setParams);
            if (SUCCESS.equals(result)) {
                ret = identifier;
                break;
            }

            try {
                Thread.sleep(2);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        jedisClient.quit();
        jedisClient.close();
    }

    return ret;
}

/*
 * @param lockName    Lock name
 * @param identifier  Lock ID
 */
public void releaseLock(String lockName, String identifier) {
    Jedis jedisClient = new Jedis(host, port);

    try {
        String authMsg = jedisClient.auth("passwd");
        if (!SUCCESS.equals(authMsg)) {
            System.out.println("AUTH FAILED: " + authMsg);
        }

        String lockKey = "DLock:" + lockName;
        if (identifier.equals(jedisClient.get(lockKey))) {
            jedisClient.del(lockKey);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        jedisClient.quit();
        jedisClient.close();
    }
}

```

```
}  
}  
}
```

### NOTICE

The code only shows how DCS implements access control using locks. During actual implementation, deadlock and lock check also need to be considered.

Assume that 20 threads are used to seckill ten Mate 10 mobile phones. The content of the test class **CaseTest.java** is as follows:

```
package dcsDemo01;  
import java.util.UUID;  
  
public class CaseTest {  
    public static void main(String[] args) {  
        ServiceOrder service = new ServiceOrder();  
        for (int i = 0; i < 20; i++) {  
            ThreadBuy client = new ThreadBuy(service);  
            client.start();  
        }  
    }  
}  
  
class ServiceOrder {  
    private final int MAX = 10;  
  
    DistributedLock DLock = new DistributedLock();  
  
    int n = 10;  
  
    public void handleOder() {  
        String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();  
        String identifier = DLock.getLockWithTimeout("Mate 10", 10000, 2000);  
        System.out.println("Processing order for user " + userName + "");  
        if(n > 0) {  
            int num = MAX - n + 1;  
            System.out.println("User "+ userName + " is allocated number " + num + " mobile phone. Number  
of mobile phones left: " + (--n) + "");  
        }else {  
            System.out.println("User "+ userName + " order failed.");  
        }  
        DLock.releaseLock("Mate 10", identifier);  
    }  
}  
  
class ThreadBuy extends Thread {  
    private ServiceOrder service;  
  
    public ThreadBuy(ServiceOrder service) {  
        this.service = service;  
    }  
  
    @Override  
    public void run() {  
        service.handleOder();  
    }  
}
```

**Step 2** Configure the connection address, port number, and password of the DCS instance in the example code file **DistributedLock.java**.

In **DistributedLock.java**, set **host** and **port** to the connection address and port number of the instance. In the **getLockWithTimeout** and **releaseLock** methods, set **passwd** to the instance access password.

**Step 3** Comment out the lock part in the test class **CaseTest**. The following is an example:

```
//The lock code is commented out in the test class:
public void handleOrder() {
    String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
    //Lock code
    //String identifier = DLock.getLockWithTimeout("Mate 10", 10000, 2000);
    System.out.println("Processing order for user " + userName + "");
    if(n > 0) {
        int num = MAX - n + 1;
        System.out.println("User " + userName + " is allocated number " + num + " mobile phone. Number of
mobile phones left: " + (--n) + "");
    }else {
        System.out.println("User " + userName + " order failed.");
    }
    //Lock code
    //DLock.releaseLock("Mate 10", identifier);
}
```

**Step 4** Compile and run a lock-free class. The purchases are disordered, as shown in the following:

```
Processing order for user e04934ddThread-5
Processing order for user a4554180Thread-0
User a4554180Thread-0 is allocated number 2 mobile phone. Number of mobile phones left: 8.
Processing order for user b58eb811Thread-10
User b58eb811Thread-10 is allocated number 3 mobile phone. Number of mobile phones left: 7.
Processing order for user e8391c0eThread-19
Processing order for user 21fd133aThread-13
Processing order for user 1dd04ff4Thread-6
User 1dd04ff4Thread-6 is allocated number 6 mobile phone. Number of mobile phones left: 4.
Processing order for user e5977112Thread-3
Processing order for user 4d7a8a2bThread-4
User e5977112Thread-3 is allocated number 7 mobile phone. Number of mobile phones left: 3.
Processing order for user 18967410Thread-15
User 18967410Thread-15 is allocated number 9 mobile phone. Number of mobile phones left: 1.
Processing order for user e4f51568Thread-14
User 21fd133aThread-13 is allocated number 5 mobile phone. Number of mobile phones left: 5.
User e8391c0eThread-19 is allocated number 4 mobile phone. Number of mobile phones left: 6.
Processing order for user d895d3f1Thread-12
User d895d3f1Thread-12 order failed.
Processing order for user 7b8d2526Thread-11
User 7b8d2526Thread-11 order failed.
Processing order for user d7ca1779Thread-8
User d7ca1779Thread-8 order failed.
Processing order for user 74fca0ecThread-1
User 74fca0ecThread-1 order failed.
User e04934ddThread-5 is allocated number 1 mobile phone. Number of mobile phones left: 9.
User e4f51568Thread-14 is allocated number 10 mobile phone. Number of mobile phones left: 0.
Processing order for user aae76a83Thread-7
User aae76a83Thread-7 order failed.
Processing order for user c638d2cfThread-2
User c638d2cfThread-2 order failed.
Processing order for user 2de29a4eThread-17
User 2de29a4eThread-17 order failed.
Processing order for user 40a46ba0Thread-18
User 40a46ba0Thread-18 order failed.
Processing order for user 211fd9c7Thread-9
User 211fd9c7Thread-9 order failed.
Processing order for user 911b83fcThread-16
User 911b83fcThread-16 order failed.
User 4d7a8a2bThread-4 is allocated number 8 mobile phone. Number of mobile phones left: 2.
```

**Step 5** Add the lock code back to **CaseTest**, and compile and run the code. The following shows sequential purchases:

```
Processing order for user eee56fb7Thread-16
User eee56fb7Thread-16 is allocated number 1 mobile phone. Number of mobile phones left: 9.
Processing order for user d6521816Thread-2
User d6521816Thread-2 is allocated number 2 mobile phone. Number of mobile phones left: 8.
```

```
Processing order for user d7b3b983Thread-19
User d7b3b983Thread-19 is allocated number 3 mobile phone. Number of mobile phones left: 7.
Processing order for user 36a6b97aThread-15
User 36a6b97aThread-15 is allocated number 4 mobile phone. Number of mobile phones left: 6.
Processing order for user 9a973456Thread-1
User 9a973456Thread-1 is allocated number 5 mobile phone. Number of mobile phones left: 5.
Processing order for user 03f1de9aThread-14
User 03f1de9aThread-14 is allocated number 6 mobile phone. Number of mobile phones left: 4.
Processing order for user 2c315ee6Thread-11
User 2c315ee6Thread-11 is allocated number 7 mobile phone. Number of mobile phones left: 3.
Processing order for user 2b03b7c0Thread-12
User 2b03b7c0Thread-12 is allocated number 8 mobile phone. Number of mobile phones left: 2.
Processing order for user 75f25749Thread-0
User 75f25749Thread-0 is allocated number 9 mobile phone. Number of mobile phones left: 1.
Processing order for user 26c71db5Thread-18
User 26c71db5Thread-18 is allocated number 10 mobile phone. Number of mobile phones left: 0.
Processing order for user c32654dbThread-17
User c32654dbThread-17 order failed.
Processing order for user df94370aThread-7
User df94370aThread-7 order failed.
Processing order for user 0af94cddThread-5
User 0af94cddThread-5 order failed.
Processing order for user e52428a4Thread-13
User e52428a4Thread-13 order failed.
Processing order for user 46f91208Thread-10
User 46f91208Thread-10 order failed.
Processing order for user e0ca87bbThread-9
User e0ca87bbThread-9 order failed.
Processing order for user f385af9aThread-8
User f385af9aThread-8 order failed.
Processing order for user 46c5f498Thread-6
User 46c5f498Thread-6 order failed.
Processing order for user 935e0f50Thread-3
User 935e0f50Thread-3 order failed.
Processing order for user d3eaae29Thread-4
User d3eaae29Thread-4 order failed.
```

----End

## 4.1.2 Ranking with DCS

### Overview

Ranking is a function commonly used on web pages and apps. It is implemented by listing key-values in descending order. However, a huge number of concurrent operation and query requests can result in a performance bottleneck, significantly increasing latency.

Ranking using DCS for Redis provides the following advantages:

- Data is stored in the memory, so read/write is fast.
- Multiple types of data structures, such as strings, lists, sets, and hashes are supported.

### Prerequisites

- A DCS instance has been created, and is in the **Running** state.
- The network between the client server and the DCS instance is connected:
  - When the client and the DCS Redis instance are in the same VPC:  
By default, networks in a VPC can communicate with each other.
  - When the client and the DCS Redis instance are in different VPCs in the same region:

If the client and DCS Redis instance are not in the same VPC, connect them by establishing a VPC peering connection. For details, see "Does DCS Support Cross-VPC Access?" in *Distributed Cache Service User Guide* > FAQs.

- To access a Redis instance of another region on a client

If the client server and the Redis instance are not in the same region, connect the network using Direct Connect. For details, see *Direct Connect User Guide*.

- You have installed **JDK1.8** (or later) and a development tool (**Eclipse** is used as an example) on the client server, and downloaded the **Jedis client**.

The development tools and clients mentioned in this document are for example only.

## Procedure

**Step 1** Run Eclipse on the server. Choose **File > New Project** to create a Java project named **dcsDemo02**.

**Step 2** Choose **New > Class** to create a **productSalesRankDemo.java** file.

**Step 3** Copy the following demo code to the **productSalesRankDemo.java** file.

```
package dcsDemo02;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.UUID;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;

public class productSalesRankDemo {
    static final int PRODUCT_KINDS = 30;

    public static void main(String[] args) {
        // Address and port for connecting to the Redis instance. Replace them with the actual values.
        String host = "192.168.0.246";
        int port = 6379;

        Jedis jedisClient = new Jedis(host, port);

        try {
            // Password for connecting to the Redis instance. Replace it with the actual value.
            String authMsg = jedisClient.auth("*****");
            if (!authMsg.equals("OK")) {
                System.out.println("AUTH FAILED: " + authMsg);
            }
        }

        //Key
        String key = "Best-seller Rankings";

        jedisClient.del(key);

        //Generate product data at random
        List<String> productList = new ArrayList<>();
        for(int i = 0; i < PRODUCT_KINDS; i++) {
            productList.add("product-" + UUID.randomUUID().toString());
        }

        //Generate sales volume at random
        for(int i = 0; i < productList.size(); i++) {
            int sales = (int)(Math.random() * 20000);
```



```
        String product = productList.get(i);
        //Insert sales volume into Redis SortedSet
        jedisClient.zadd(key, sales, product);
    }

    System.out.println();
    System.out.println("          "+key);

    //Obtain all lists and display the lists by sales volume
    Set<Tuple> sortedProductList = jedisClient.zrevrangeWithScores(key, 0, -1);
    for(Tuple product : sortedProductList) {
        System.out.println("Product ID: " + product.getElement() + ", Sales volume: "
            + Double.valueOf(product.getScore()).intValue());
    }

    System.out.println();
    System.out.println("          "+key);
    System.out.println("          Top 5 Best-sellers");

    //Obtain the top 5 best-selling products and display the products by sales volume
    Set<Tuple> sortedTopList = jedisClient.zrevrangeWithScores(key, 0, 4);
    for(Tuple product : sortedTopList) {
        System.out.println("Product ID: " + product.getElement() + ", Sales volume: "
            + Double.valueOf(product.getScore()).intValue());
    }
}
catch (Exception e) {
    e.printStackTrace();
}
finally {
    jedisClient.quit();
    jedisClient.close();
}
}
```

**Step 4** Configure the connection address, port, and password for the DCS instance in the example code file.

**Step 5** Compile and run the code.

----End

## Operation Result

Compile and run the preceding Demo code. The operation result is as follows:

```
Best-seller Rankings
Product ID: product-b290c0d4-e919-4266-8eb5-7ab84b19862d, Sales volume: 18433
Product ID: product-e61a0642-d34f-46f4-a720-ee35940a5e7f, Sales volume: 18334
Product ID: product-ceeab7c3-69a7-4994-afc6-41b7bc463d44, Sales volume: 18196
Product ID: product-f2bdc549-8b3e-4db1-8cd4-a2ddef4f5d97, Sales volume: 17870
Product ID: product-f50ca2de-7fa4-45a3-bf32-23d34ac15a41, Sales volume: 17842
Product ID: product-d0c364e0-66ec-48a8-9ac9-4fb58adfd033, Sales volume: 17782
Product ID: product-5e406bbf-47c7-44a9-965e-e1e9b62ed1cc, Sales volume: 17093
Product ID: product-0c4d31ee-bb15-4c88-b319-a69f74e3c493, Sales volume: 16432
Product ID: product-a986e3a4-4023-4e00-8104-db97e459f958, Sales volume: 16380
Product ID: product-a3ac9738-bed2-4a9c-b96a-d8511ae7f03a, Sales volume: 15305
Product ID: product-6b8ad4b7-e134-480f-b3ae-3d35d242cb53, Sales volume: 14534
Product ID: product-26a9b41b-96b1-4de0-932b-f78d95d55b2d, Sales volume: 11417
Product ID: product-1f043255-a1f9-40a0-b48b-f40a81d07e0e, Sales volume: 10875
Product ID: product-c8fee24c-d601-4e0e-9d18-046a65e59835, Sales volume: 10521
Product ID: product-5869622b-1894-4702-b750-d76ff4b29163, Sales volume: 10271
Product ID: product-ff0317d2-d7be-4021-9d25-1f997d622768, Sales volume: 9909
Product ID: product-da254e81-6dec-4c76-928d-9a879a11ed8d, Sales volume: 9504
Product ID: product-fa976c02-b175-4e82-b53a-8c0df96fe877, Sales volume: 8630
Product ID: product-0624a180-4914-46b9-84d0-9dfbbdaa0da2, Sales volume: 8405
```

Product ID: product-d0079955-eaea-47b2-845f-5ff05a110a70, Sales volume: 7930  
 Product ID: product-a53145ef-1db9-4c4d-a029-9324e7f728fe, Sales volume: 7429  
 Product ID: product-9b1a1fd1-7c3b-4ae8-9fd3-ab6a0bf71cae, Sales volume: 5944  
 Product ID: product-cf894aee-c1cb-425e-a644-87ff06485eb7, Sales volume: 5252  
 Product ID: product-8bd78ba8-f2c4-4e5e-b393-60aa738eceeae, Sales volume: 4903  
 Product ID: product-89b64402-c624-4cf1-8532-ae1b4ec4cab, Sales volume: 4527  
 Product ID: product-98b85168-9226-43d9-b3cf-ef84e1c3d75f, Sales volume: 3095  
 Product ID: product-0dda314f-22a7-464b-ab8c-2f8f00823a39, Sales volume: 2425  
 Product ID: product-de7eb085-9435-4924-b6fa-9e9fe552d5a7, Sales volume: 1694  
 Product ID: product-9beadc07-aab0-438c-ac5e-bcc72b9d9c36, Sales volume: 1135  
 Product ID: product-43834316-4aca-4fb2-8d2d-c768513015c5, Sales volume: 256

Best-seller Rankings  
Top 5 Best-sellers

Product ID: product-b290c0d4-e919-4266-8eb5-7ab84b19862d, Sales volume: 18433  
 Product ID: product-e61a0642-d34f-46f4-a720-ee35940a5e7f, Sales volume: 18334  
 Product ID: product-ceeab7c3-69a7-4994-afc6-41b7bc463d44, Sales volume: 18196  
 Product ID: product-f2bdc549-8b3e-4db1-8cd4-a2ddef4f5d97, Sales volume: 17870  
 Product ID: product-f50ca2de-7fa4-45a3-bf32-23d34ac15a41, Sales volume: 17842

## 4.2 Network Connection

### 4.2.1 Configuring Redis Client Retry

#### Importance of Retry

Both the client and server may encounter temporary faults (such as transient network or disk jitter, service unavailability, or invoking timeout, due to infrastructure or running environment reasons). As a result, Redis operations may fail. You can design automated retry mechanisms to reduce the impact of such faults and ensure successful execution.

#### Scenarios Where Redis Operations Fail

Scenario	Description
Master/standby switchover triggered by a fault	If the master node is faulty due to Redis underlying hardware or other reasons, a master/standby switchover is triggered to ensure that the instance is still available. A master/standby switchover causes instance disconnection for 15 to 30s:
Read-only during specification modification	During specification modification, the instance may be disconnected for seconds and read-only for minutes.
Request blockage caused by slow queries	Operations whose time complexity is $O(N)$ cause slow queries and request blockage. In this case, other client requests may temporarily fail.
Complex network environment	Due to the complex network environment between the client and the Redis server, network jitter, packet loss, and data retransmission may occur occasionally. In this case, client requests may temporarily fail.

Scenario	Description
Complex hardware issues	Client requests may temporarily fail due to occasional hardware faults, such as VM HA and disk latency jitter.

## Recommended Retry Rules

Retry Rule	Description
Retry only idempotent operations.	<p>Timeout may occur in any of the following phases:</p> <ul style="list-style-type: none"> <li>• A command is successfully sent by the client but has not reached Redis.</li> <li>• The command has reached Redis, but the execution times out.</li> <li>• Redis has executed the command, but the result returned to the client times out.</li> </ul> <p>A retried operation may be repeatedly executed in Redis. Therefore, not all operations are suitable to be retried. You are advised to retry only idempotent operations, such as running the <b>SET</b> command. For example, if you run the <b>SET a b</b> command multiple times, the value of <b>a</b> can only be <b>b</b> or the execution fails. If you run <b>LPUSH mylist a</b>, which is not idempotent, <b>mylist</b> may contain multiple <b>a</b> elements.</p>
Configure proper retry times and interval.	<p>Configure the retry times and interval based on service requirements in actual scenarios to prevent the following problems:</p> <ul style="list-style-type: none"> <li>• If the number of retries is insufficient or the interval is too long, the application may fail to complete operations.</li> <li>• If the number of retries is too large or the interval is too short, the application may occupy too many system resources and the server may be blocked due to too many requests.</li> </ul> <p>Common retry interval policies include immediate retry, fixed-interval retry, exponential backoff retry, and random backoff retry.</p>
Avoid retry nesting.	Retry nesting may cause the retry interval to be exponentially amplified.
Record retry exceptions and print failure reports.	During retry, you can print retry error logs at the WARN level.

## Jedis Client Retry Configurations

- Retries are not supported in native JedisPool mode (for single-node, master/standby, and Proxy Cluster instances). However, you can implement retries by referring to [JedisClusterCommand](#).
- Retries are supported in JedisCluster mode. You can set the **maxAttempts** parameter to define the number of retry times when a failure occurs. The default value is **5**. By default, all JedisCluster operations invoke the retry method.

Example code:

```
@Bean
JedisCluster jedisCluster() {
    Set<HostAndPort> hostAndPortsSet = new HashSet<>();
    hostAndPortsSet.add(new HostAndPort("{dcs_instance_address}", 6379));
    JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
    jedisPoolConfig.setMaxIdle(100);
    jedisPoolConfig.setMinIdle(1);
    jedisPoolConfig.setMaxTotal(1000);
    jedisPoolConfig.setMaxWaitMillis(2000);
    jedisPoolConfig.setMaxAttempts(5);
    return new JedisCluster(hostAndPortsSet, jedisPoolConfig);
}
```

**Table 4-1** Recommended Jedis connection pool parameter settings

Parameter	Description	Recommended Setting
maxTotal	Maximum number of connections	<p>Set this parameter based on the number of HTTP threads of the web container and reserved connections. Assume that the <b>maxConnections</b> parameter of the Tomcat Connector is set to <b>150</b> and each HTTP request may concurrently send two requests to Redis, you are advised to set this parameter to at least 400 (<math>150 \times 2 + 100</math>).</p> <p><b>Limit:</b> The value of <b>maxTotal</b> multiplied by the number of client nodes (CCE containers or service VMs) must be less than the maximum number of connections allowed for a single DCS Redis instance.</p> <p>For example, if <b>maxClients</b> of a master/standby DCS Redis instance is 10,000 and <b>maxTotal</b> of a single client is 500, the maximum number of clients is 20.</p>
maxIdle	Maximum number of idle connections	Use the same configuration as <b>maxTotal</b> .

Parameter	Description	Recommended Setting
minIdle	Minimum number of idle connections	<p>Generally, you are advised to set this parameter to 1/X of <b>maxTotal</b>. For example, the recommended value is <b>100</b>.</p> <p>In performance-sensitive scenarios, you can set this parameter to the value of <b>maxIdle</b> to prevent the impact caused by frequent connection quantity changes. For example, set this parameter to <b>400</b>.</p>
maxWaitMillis	Maximum waiting time for obtaining a connection, in milliseconds	<p>The recommended maximum waiting time for obtaining a connection from the connection pool is the maximum tolerable timeout of a single service minus the timeout for command execution. For example, if the maximum tolerable HTTP failure is 15s and the timeout of Redis requests is 10s, set this parameter to 5s.</p>
timeout	Command execution timeout, in milliseconds	<p>This parameter indicates the maximum timeout for running a Redis command. Set this parameter based on the service logic. You are advised to set this timeout to at least 210 ms to ensure network fault tolerance. For special detection logic or environment exception detection, you can adjust this timeout to seconds.</p>

Parameter	Description	Recommended Setting
minEvictableIdleTimeMillis	Idle connection eviction time, in milliseconds. If a connection is not used for a period longer than this, it will be released.	If you do not want the system to frequently re-establish disconnected connections, set this parameter to a large value (xx minutes) or set this parameter to <b>-1</b> and check idle connections periodically.
timeBetweenEvictionRunsMillis	Interval for detecting idle connections, in milliseconds	The value is estimated based on the number of idle connections in the system. For example, if this interval is set to 30s, the system detects connections every 30s. If an abnormal connection is detected within 30s, it will be removed. Set this parameter based on the number of connections. If the number of connections is too large and this interval is too short, request resources will be wasted. If there are hundreds of connections, you are advised to set this parameter to 30s. The value can be dynamically adjusted based on system requirements.
testOnBorrow	Indicates whether to check the connection validity using the <b>ping</b> command when borrowing connections from the resource pool. Invalid connections will be removed.	If your service is extremely sensitive to connections and the performance is acceptable, you can set this parameter to <b>True</b> . Generally, you are advised to set this parameter to <b>False</b> to enable idle connection detection.

Parameter	Description	Recommended Setting
testWhileIdle	Indicates whether to use the <b>ping</b> command to monitor the connection validity during idle resource monitoring. Invalid connections will be destroyed.	True
testOnReturn	Indicates whether to check the connection validity using the <b>ping</b> command when returning connections to the resource pool. Invalid connections will be removed.	False
maxAttempts	Number of connection retries when JedisCluster is used	Recommended value: 3–5. Default value: <b>5</b> . Set this parameter based on the maximum timeout intervals of service APIs and a single request. The maximum value is <b>10</b> . If the value exceeds <b>10</b> , the processing time of a single request is too long, blocking other requests.

## 4.3 Usage Guide

### 4.3.1 Suggestions on Using DCS

#### Service Usage

Principle	Description	Remarks
Separate hot data from cold data.	You can store frequently accessed data (hot data) in Redis, and infrequently accessed data (cold data) in databases such as MySQL and Elasticsearch.	Infrequently accessed data stored in the memory occupies Redis space and does not accelerate access.



Principle	Description	Remarks
Differentiate service data.	Store unrelated service data in different Redis instances.	This prevents services from affecting each other and prevents single instances from being too large. This also enables you to quickly restore services in case of faults.
	Do not use the <b>SELECT</b> command for multi-DB on a single instance.	Multi-DB on a single Redis instance does not provide good isolation and is no longer in active development by open-source Redis. You are advised not to depend on this feature in the future.
Set a proper eviction policy.	If the eviction policy is set properly, Redis can still function when the memory is used up unexpectedly.	You can that meets your service requirements. The default eviction policy used by DCS is <b>volatile-lru</b> .
Use Redis as cache.	Do not over-rely on Redis transactions.	After a transaction is executed, it cannot be rolled back.
	If data is abnormal, clear the cache for data restoration.	Redis does not have a mechanism or protocol to ensure strong data consistency. Therefore, services cannot over-rely on the accuracy of Redis data.
	When using Redis as cache, set expiration on all keys. Do not use Redis as a database.	Set expiration as required, but a longer expiration is not necessarily better.
Prevent cache breakdown.	Use Redis together with local cache. Store frequently used data in the local cache and regularly update it asynchronously.	-
Prevent cache penetration.	Non-critical path operations are passed through to the database. Limit the rate of access to the database.	-

Principle	Description	Remarks
Do not use Redis as a message queue.	In pub/sub scenarios, do not use Redis as a message queue.	<ul style="list-style-type: none"> <li>• Unless otherwise required, you are not advised to use Redis as a message queue.</li> <li>• Using Redis as a message queue causes capacity, network, performance, and function issues.</li> <li>• If message queues are required, use Kafka for throughput and RocketMQ for reliability.</li> </ul>
Select proper specifications.	If service growth causes increases in Redis requests, use Proxy Cluster or Redis Cluster instances.	Scaling up single-node and master/standby instances only expands the memory and bandwidth, but cannot enhance the computing capabilities.
	In production, do not use single-node instances. Use master/standby or cluster instances.	-
	Do not use large specifications for master/standby instances.	Redis forks a process when rewriting AOF or running the <b>BGSAVE</b> command. If the memory is too large, responses will be slow.
Prepare for degradation or disaster recovery.	When a cache miss occurs, data is obtained from the database. Alternatively, when a fault occurs, allow another Redis to take over services automatically.	-

## Data Design

Category	Principle	Description	Remarks
Keys	Keep the format consistent.	Use the service name or database name as the prefix, followed by colons (:). Ensure that key names have clear meanings.	For example: <i>service name.sub-service name.ID</i> .
	Minimize the key length.	Minimize the key length without compromising clarity of the meaning. Abbreviate common words. For example, <b>user</b> can be abbreviated to <b>u</b> , and <b>messages</b> can be abbreviated to <b>msg</b> .	Use up to 128 bytes. The shorter the better.
	Do not use special characters except braces ({}).	Do not use special characters such as spaces, line brakes, single or double quotation marks, and other escape characters.	Redis uses braces ({} ) to signify hash tags. Braces in key names must be used correctly to avoid unbalanced shards.
Values	Use appropriate value sizes.	Keep the value of a key within 10 KB.	Large values may cause unbalanced shards, hot keys, traffic or CPU usage surges, and scaling or migration failures. These problems can be avoided by proper design.

Category	Principle	Description	Remarks
	Use appropriate number of elements in each key.	Do not include too many elements in each Hash, Set, or List. It is recommended that each key contain up to 5000 elements.	Time complexity of some commands, such as <b>HGETALL</b> , is directly related to the quantity of elements in a key. If commands whose time complexity is $O(N)$ or higher are frequently executed and a key has a large number of elements, there may be slow requests, unbalanced shards, or hot keys.
	Use appropriate data types.	This saves memory and bandwidth.	For example, to store multiple attributes of a user, you can use multiple keys, such as <b>set u:1:name "X"</b> and <b>set u:1:age 20</b> . To save memory usage, you can also use the <b>HMSET</b> command to set multiple fields to their respective values in the hash stored at one key.
	Set appropriate timeout.	Do not set a large number of keys to expire at the same time.	When setting key expiration, add or subtract a random offset from a base expiry time, to prevent a large number of keys from expiring at the same time. Otherwise, CPU usage will be high at the expiry time.

## Command Usage

Principle	Description	Remarks
Exercise caution when using commands with time complexity of $O(N)$ .	Pay attention to the value of $N$ for commands whose time complexity is $O(N)$ . If the value of $N$ is too large, Redis will be blocked and the CPU usage will be high.	For example, the <b>HGETALL</b> , <b>LRange</b> , <b>SMembers</b> , <b>ZRange</b> , and <b>SInter</b> commands will consume a large number of CPU resources if there is a large number of elements. Alternatively, you can use <b>SCAN</b> sister commands, such as <b>HSCAN</b> , <b>SSCAN</b> , and <b>ZSCAN</b> commands.
Do not use high-risk commands.	Do not use high-risk commands such as <b>FLUSHALL</b> , <b>KEYS</b> , and <b>HGETALL</b> , or rename them.	For details, see section "Renaming Commands" in the <i>User Guide</i> .
Exercise caution when using the <b>SELECT</b> command.	Redis does not have a strong support for multi-DB. Redis is single-threaded, so databases interfere with each other. You are advised to use multiple Redis instances instead of using multi-DB on one instance.	-
Use batch operations to improve efficiency.	For batch operations, use the <b>MGET</b> command, <b>MSET</b> command, or pipelining to improve efficiency, but do not include a large number of elements in one batch operation.	<p><b>MGET</b> command, <b>MSET</b> command, and pipelining differ in the following ways:</p> <ul style="list-style-type: none"> <li>• <b>MGET</b> and <b>MSET</b> are atomic operations, while pipelining is not.</li> <li>• Pipelining can be used to send multiple commands at a time, while <b>MGET</b> and <b>MSET</b> cannot.</li> <li>• Pipelining must be supported by both the server and the client.</li> </ul>
Do not use time-consuming code in Lua scripts.	The timeout of Lua scripts is 5s, so avoid using long scripts.	Long scripts: time-consuming sleep statements or long loops.

Principle	Description	Remarks
Do not use random functions in Lua scripts.	When invoking a Lua script, do not use random functions to specify keys. Otherwise, the execution results will be inconsistent between the master and standby nodes, causing data inconsistency.	-
Follow the rules for using Lua on cluster instances.	Follow the rules for using Lua on cluster instances.	<ul style="list-style-type: none"> <li>• When the <b>EVAL</b> or <b>EVALSHA</b> command is run, the command parameter must contain at least one key. Otherwise, the client displays the error message "ERR eval/evalsha numkeys must be bigger than zero in redis cluster mode."</li> <li>• When the <b>EVAL</b> or <b>EVALSHA</b> command is run, a cluster DCS Redis instance uses the first key to compute slots. Ensure that the keys to be operated are in the same slot.</li> </ul>
Optimize multi-key operation commands such as <b>MGET</b> and <b>HMGET</b> with parallel processing and non-blocking I/O.	Some clients do not treat these commands differently. Keys in such a command are processed sequentially before their values are returned in a batch. This process is slow and can be optimized through pipelining.	For example, running the <b>MGET</b> command on a cluster using Lettuce is dozens of times faster than using Jedis, because Lettuce uses pipelining and non-blocking I/O while Jedis does not have a special plan itself. To use Jedis in such scenarios, you need to implement slot grouping and pipelining by yourself.

Principle	Description	Remarks
Do not use the <b>DEL</b> command to directly delete big keys.	Deleting big keys, especially Sets, using <b>DEL</b> blocks other requests.	<p>In Redis 4.0 and later, you can use the <b>UNLINK</b> command to delete big keys safely. This command is non-blocking.</p> <p>In versions earlier than Redis 4.0:</p> <ul style="list-style-type: none"> <li>• To delete big Hashes, use <b>HSCAN + HDEL</b> commands.</li> <li>• To delete big Lists, use the <b>LTRIM</b> command.</li> <li>• To delete big Sets, use <b>SSCAN + SREM</b> commands.</li> <li>• To delete big Sorted Sets, use <b>ZSCAN + ZREM</b> commands.</li> </ul>

## SDK Usage

Principle	Description	Remarks
Use connection pools and persistent connections ("pconnect" in Redis terminology).	The performance of short connections ("connect" in Redis terminology) is poor. Use clients with connection pools.	Frequently connecting to and disconnecting from Redis will unnecessarily consume a lot of system resources and can cause host breakdown in extreme cases. Ensure that the Redis client connection pool is correctly configured.
The client must perform fault tolerance in case of faults or slow requests.	The client should have fault tolerance and retry mechanisms in case of master/standby switchover, command timeout, or slow requests caused by network fluctuation or configuration errors.	See <a href="#">Configuring Redis Client Retry</a> .

Principle	Description	Remarks
Set appropriate interval and number of retries.	Do not set the retry interval too short or too long.	<ul style="list-style-type: none"> <li>If the retry interval is very short, for example, shorter than 200 milliseconds, a retry storm may occur, and can easily cause service avalanche.</li> <li>If the retry interval is very long or the number of retries is set to a large value, the service recovery may be slow in the case of a master/standby switchover.</li> </ul>
Avoid using Lettuce.	Lettuce is the default client of Spring and stands out in terms of performance. However, Jedis is more stable because it is better at detecting and handling connection errors and network fluctuations. Therefore, Jedis is recommended.	<p>Lettuce has the following problems:</p> <ul style="list-style-type: none"> <li>By default, Lettuce does not have cluster topology update configurations. When the cluster topology changes (for example after a master/standby switchover or scaling), new nodes cannot be identified, causing service failures.</li> <li>Lettuce cannot validate connections in the connection pool. If an invalid connection is used, services will fail and may become unavailable in minutes.</li> </ul>

## O&M and Management

Principle	Description	Remarks
Use passwords in production.	In production systems, use passwords to protect Redis.	-
Ensure security on the live network.	Do not allow unauthorized developers to connect to redis-server in the production environment.	-



Principle	Description	Remarks
Verify the fault handling capability or disaster recovery logic of the service.	Organize drills in the test environment or pre-production environment to verify service reliability in Redis master/standby switchover, breakdown, or scaling scenarios.	Master/standby switchover can be triggered manually on the console. It is strongly recommended that you use Lettuce for these drills.
Configure monitoring.	Pay attention to the Redis capacity and expand it before overload.	Configure CPU, memory, and bandwidth alarms based on the alarm thresholds.
Perform routine health checks.	Perform routine checks on the memory usage of each node and whether the memory usage of the master nodes is balanced.	If memory usage is unbalanced, big keys exist and need to be split and optimized.
	Perform routine analysis on hot keys and check whether there are frequently accessed keys.	-
	Perform routine diagnosis on Redis commands and check whether O(N) commands have potential risks.	Even if an O(N) command is not time-consuming, it is recommended that R&D engineers analyze whether the value of N will increase with service growth.
	Perform routine analysis on slow query logs.	Detect potential risks based on slow query logs and rectify faults as soon as possible.

# 5 FAQs

## 5.1 Instance Types/Versions

### 5.1.1 Comparing Versions

When creating a DCS Redis instance, you can select the cache engine version and the instance type.

- **Version**

DCS supports Redis 3.0/4.0/5.0/6.0. [Table 5-1](#) describes the differences between these versions. For more information about Redis features, see [New Features of DCS for Redis 4.0](#), [New Features of DCS for Redis 6.0](#), and [New Features of DCS for Redis 5.0](#).

**Table 5-1** Differences between Redis versions

Item	Redis 3.0	Redis 4.0 and Redis 5.0	Redis 6.0
Open-source compatibility	Redis 3.0.7	Redis 4.0.14 The latest DCS for Redis 5.0 is compatible with Redis 5.0.14. For existing users, see <a href="#">How Do I View the Version of a DCS Redis Instance?</a>	6.2.7
Instance deployment mode	Based on VMs	Containerized based on physical servers	Containerized based on physical servers

Item	Redis 3.0	Redis 4.0 and Redis 5.0	Redis 6.0
CPU architecture	x86 and Arm	x86 and Arm	x86
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds	8 seconds
QPS	50,000 QPS per node	50,000 QPS per node	150,000 QPS per node
Domain name connection	Supported within a VPC	Supported within a VPC	Supported within a VPC
Visualized data management	Not supported	Web CLI for connecting to Redis and managing data	Web CLI for connecting to Redis and managing data
Instance types	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby, Proxy Cluster, Redis Cluster	Single-node, master/standby, Redis Cluster
Instance total memory	Ranges from 2 GB to 1024 GB.	Regular specifications range from 2 GB to 1024 GB. Small specifications of 128 MB, 256 MB, 512 MB, and 1 GB are also available for single-node and master/standby instances.	4 GB, 8 GB, 16 GB, 32 GB, and 64 GB (128 MB, 256 MB, 512 MB, and 1 GB are additionally supported for single-node and master/standby instances)

Item	Redis 3.0	Redis 4.0 and Redis 5.0	Redis 6.0
Scale-up or scale-down	Online scale-up and scale-down	Online scale-up and scale-down	Online scale-up and scale-down
Backup and restoration	Supported for master/standby and cluster instances	Supported for master/standby and cluster instances	Master/Standby, Redis Cluster

 **NOTE**

The underlying architectures vary by Redis version. Once a Redis version is chosen, it cannot be changed. For example, you cannot upgrade a DCS Redis 3.0 instance to Redis 4.0 or 5.0. If you require a higher Redis version, create a new instance that meets your requirements and then migrate data from the old instance to the new one.

DCS Redis 3.0 instances have been taken offline at new sites, but can still be used at existing sites. You are advised to use DCS Redis 4.0 and later instances.

- **Instance type**

DCS provides single-node, master/standby, Proxy Cluster, and Redis Cluster instance types. For details about their architectures and application scenarios, see section "DCS Instance Types".

## 5.1.2 How Do I View the Version of a DCS Redis Instance?

Connect to the instance and run the **INFO** command.

**Figure 5-1** Querying instance information

```
> INFO
# Server

redis_version:5.0.14

patch_version:5.0.14.1

redis_git_sha1:00000000

redis_git_dirty:0
```

## 5.1.3 New Features of DCS for Redis 4.0

Compared with DCS for Redis 3.0, DCS for Redis 4.0 and later versions add support for the new features of open-source Redis and supports faster instance creation.

Instance deployment changed from the VM mode to physical server-based containerization mode. An instance can be created within 8 to 10 seconds.

Redis 4.0 provides the following new features:

1. New commands, such as **MEMORY** and **SWAPDB**
2. Lazyfree, delaying the deletion of large keys and reducing the impact of the deletion on system resources
3. Memory performance optimization, that is, active defragmentation

## MEMORY Command

In Redis 3.0 and earlier versions, you can execute the **INFO MEMORY** command to learn only the limited memory statistics. Redis 4.0 introduces the **MEMORY** command to help you better understand Redis memory usage.

```
127.0.0.1:6379[8]> memory help
1) MEMORY <subcommand> arg arg ... arg. Subcommands are:
2) DOCTOR - Return memory problems reports.
3) MALLOC-STATS -- Return internal statistics report from the memory allocator.
4) PURGE -- Attempt to purge dirty pages for reclamation by the allocator.
5) STATS -- Return information about the memory usage of the server.
6) USAGE <key> [SAMPLES <count>] -- Return memory in bytes used by <key> and its value. Nested values
are sampled up to <count
> times (default: 5).
127.0.0.1:6379[8]>
```

### usage

Enter **memory usage [key]**. If the key exists, the estimated memory used by the value of the key is returned. If the key does not exist, **nil** is returned.

```
127.0.0.1:6379[8]> set dcs "DCS is an online, distributed, in-memory cache service compatible with Redis,
and Memcached."
OK
127.0.0.1:6379[8]> memory usage dcs
(integer) 141
127.0.0.1:6379[8]>
```

### NOTE

1. **usage** collects statistics on the memory usage of the value and the key, excluding the Expire memory usage of the key.  
// The following is verified based on Redis 5.0.2. Results may differ in other Redis versions.  
192.168.0.66:6379> set a "Hello, world!"  
OK  
192.168.0.66:6379> memory usage a  
(integer) 58  
192.168.0.66:6379> set abc "Hello, world!"  
OK  
192.168.0.66:6379> memory usage abc  
(integer) 60 //After the key name length changes, the memory usage also changes. This indicates that the usage statistics contain the usage of the key.  
192.168.0.66:6379> expire abc 1000000  
(integer) 1  
192.168.0.66:6379> memory usage abc  
(integer) 60 // After the expiration time is added, the memory usage remains unchanged. This indicates that the usage statistics do not contain the expire memory usage.  
192.168.0.66:6379>
2. For hashes, lists, sets, and sorted sets, the **MEMORY USAGE** command samples statistics and provides the estimated memory usage.  
Usage: **memory usage keyset samples 1000**  
*keyset* indicates the key of a set, and *1000* indicates the number of samples.

## stats

Returns the detailed memory usage of the current instance.

Usage: **memory stats**

```
127.0.0.1:6379[8]> memory stats
1) "peak.allocated"
2) (integer) 2412408
3) "total.allocated"
4) (integer) 2084720
5) "startup.allocated"
6) (integer) 824928
7) "replication.backlog"
... ..
```

The following table describes the meanings of some return items.

**Table 5-2** memory stats

Return Value	Description
peak.allocated	Peak memory allocated by the allocator during Redis instance running. It is the same as <b>used_memory_peak</b> of <b>info memory</b> .
total.allocated	The number of bytes allocated by the allocator. It is the same as <b>used_memory</b> of <b>info memory</b>
startup.allocated	Initial amount of memory consumed by Redis at startup in bytes.
replication.backlog	Size in bytes of the replication backlog. It is specified in the <b>repl-backlog-size</b> parameter. The default value is <b>1 MB</b> .
<b>clients.slaves</b>	The total size in bytes of all replicas overheads.
<b>clients.normal</b>	The total size in bytes of all clients overheads.
overhead.total	The sum of all overheads. <b>overhead.total</b> is the total memory <b>total.allocated</b> allocated by the allocator minus the actual memory used for storing data.
keys.count	The total number of keys stored across all databases in the server.
keys.bytes-per-key	Average number of bytes occupied by each key. Note that the overhead is also allocated to each key. Therefore, this value does not indicate the average key length.
dataset.bytes	Memory bytes occupied by Redis data, that is, <b>overhead.total</b> subtracted from <b>total.allocated</b>
<b>dataset.percentage</b>	The percentage of <b>dataset.bytes</b> out of the net memory usage.
peak.percentage	The percentage of peak.allocated out of <b>total.allocated</b> .

Return Value	Description
fragmentation	Memory fragmentation rate.

## doctor

Usage: **memory doctor**

If the value of **used\_memory (total.allocated)** is less than 5 MB, **MEMORY DOCTOR** considers that the memory usage is too small and does not perform further diagnosis. If any of the following conditions is met, Redis provides diagnosis results and suggestions:

1. The peak allocated memory is greater than 1.5 times of the current **total\_allocated**, that is, **peak.allocated/total.allocated** > 1.5, indicating that the memory fragmentation rate is high, and that the RSS is much larger than **used\_memory**.
2. The value of high fragmentation/fragmentation is greater than 1.4, indicating that the memory fragmentation rate is high.
3. The average memory usage of each normal client is greater than 200 KB, indicating that the pipeline may be improperly used or the Pub/Sub client does not process messages in time.
4. The average memory usage of each slave client is greater than 10 MB, indicating that the write traffic of the master is too high.

## purge

Usage: **memory purge**

Executes the **jemalloc** internal command to release the memory. The released objects include the memory that is occupied but not used by Redis processes, that is, memory fragments.

### NOTE

**MEMORY PURGE** applies only to the Redis instance that uses **jemalloc** as the allocator.

## Lazyfree

### Problem

Redis is single-thread. When a time-consuming request is executed, all requests are queued. Before the request is completed, Redis cannot respond to other requests. As a result, performance problems may occur. One of the time-consuming requests is deleting a large key.

### Principle

The Lazyfree feature of Redis 4.0 avoids the blockage caused by deleting large keys, ensuring performance and availability.

When deleting a key, Redis asynchronously releases the memory occupied by the key. The key release operation is processed in the sub-thread of the background I/O (BIO).

## Usage

### 1. Active deletion

#### – **unlink**

Similar to **DEL**, this command removes keys. If there are more than 64 elements to be deleted, the memory release operation is executed in an independent BIO thread. Therefore, the **UNLINK** command can delete a large key containing millions of elements in a short time.

#### – **flushall/flushdb**

An **ASYNC** option was added to **FLUSHALL** and **FLUSHDB** in order to let the entire dataset or a single database to be freed asynchronously.

### 2. Passive deletion: deletion of expired keys and eviction of large keys

There are four scenarios for passive deletion and each scenario corresponds to a parameter. These parameters are disabled by default.

`lazyfree-lazy-eviction no` // Whether to enable Lazyfree when the Redis memory usage reaches **maxmemory** and the eviction policy is set.

`lazyfree-lazy-expire no` // Whether to enable Lazyfree when the key with TTL is going to expire.

`lazyfree-lazy-server-del no` // An implicit **DEL** key is used when an existing key is processed.

`slave-lazy-flush no` // Perform full data synchronization for the standby node. Before loading the RDB file of the master, the standby node executes the **FLUSHALL** command to clear its own data.

#### NOTE

To enable these configurations, contact technical support.

## Other New Commands

### 1. **swapdb**

Swaps two Redis databases.

**swapdb** *dbindex1 dbindex2*

### 2. **zlexcount**

Returns the number of elements in the sorted set.

**zlexcount** *key min max*

## Memory and Performance Optimization

1. Compared to before, the same amount of data can be stored with less memory.
2. Used memory can be defragmented and gradually evicted.

### 5.1.4 New Features of DCS for Redis 5.0

DCS for Redis 5.0 is compatible with the new features of the open-source Redis 5.0, in addition to all the improvements and new commands in Redis 4.0.

## Stream Data Structure

Stream is a new data type introduced with Redis 5.0. It supports message persistence and multicast.

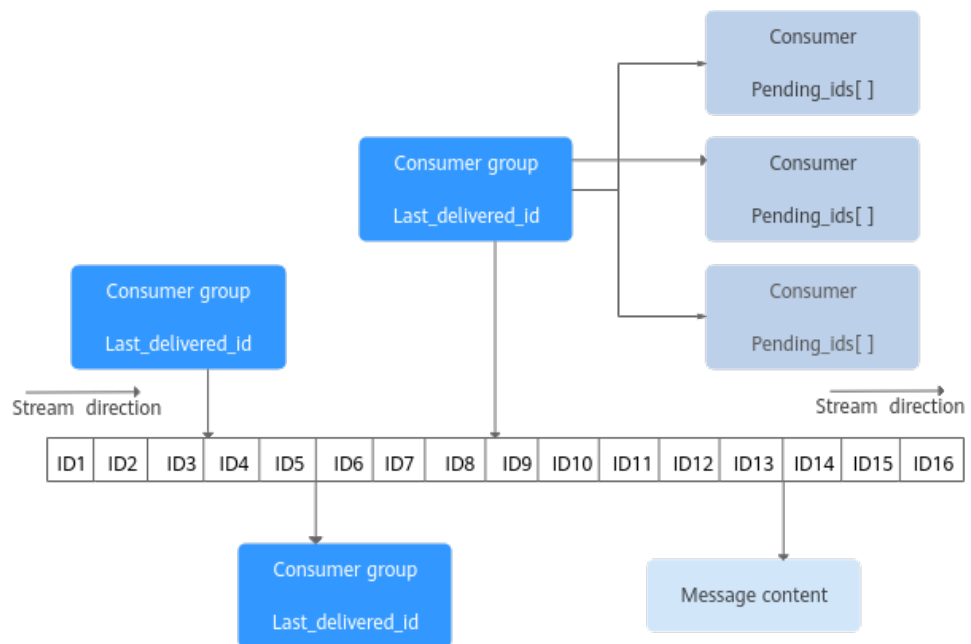
**Figure 5-2** shows the structure of a Redis stream, which allows messages to be appended to the stream.



**Streams have the following features:**

1. A stream can have multiple consumer groups.
2. Each consumer group contains a **Last\_delivered\_id** that points to the last consumed item (message) in the consumer group.
3. Each consumer group contains multiple consumers. All consumers share the **last\_delivered\_id** of the consumer group. A message can be consumed by only one consumer.
4. **pending\_ids** in the consumer can be used to record the IDs of items that have been sent to the client, but have not been acknowledged.
5. For detailed comparison between stream and other Redis data structures, see [Table 5-3](#).

**Figure 5-2** Stream data structure



**Table 5-3** Differences between streams and existing Redis data structures

Item	Stream	List, Pub/Sub, Zset
Complexity of seeking items	$O(\log(N))$	List: $O(N)$
Offset	Supported. Each item has a unique ID. The ID is not changed as other items are added or evicted.	List: Not supported. If an item is evicted, the latest item cannot be located.
Persistence	Supported. Streams are persisted to AOF and RDB files.	Pub/Sub: Not supported.

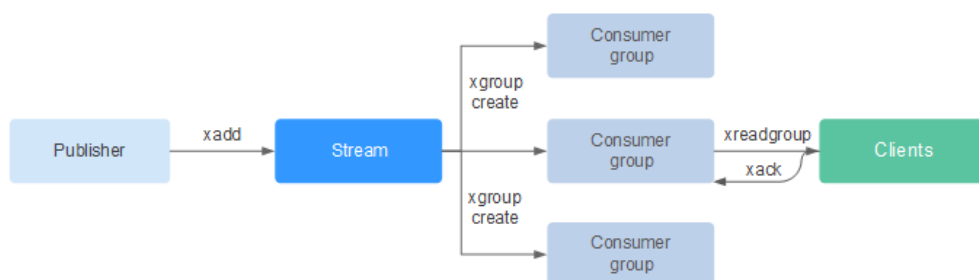
Item	Stream	List, Pub/Sub, Zset
Consumer group	Supported.	Pub/Sub: Not supported.
Acknowledgment	Supported.	Pub/Sub: Not supported.
Performance	Not related to the number of consumers.	Pub/Sub: Positively related to the number of clients.
Eviction	Streams are memory efficient by blocking to evict the data that is too old and using a radix tree and listpack.	Zset consumes more memory because it does not support inserting same items, blocking, or evicting data
Randomly deleting items	Not supported.	Zset: Supported.

### Stream commands

Stream commands are described below in the order they are used. For details, see [Table 5-4](#).

1. Run the **XADD** command to add a stream item, that is, create a stream. The maximum number of messages that can be saved can be specified when adding the item.
2. Create a consumer group by running the **XGROUP** command.
3. A consumer uses the **XREADGROUP** command to consume messages.
4. After the consumption, the client runs the **XACK** command to confirm that the consumption is successful.

**Figure 5-3** Stream commands



**Table 5-4** Stream commands description

Command	Description	Syntax
XACK	Deletes one or multiple messages from the <i>pending entry list</i> (PEL) a consumer group of the stream.	XACK key group ID [ID ...]

Command	Description	Syntax
XADD	Adds a specified entry to the stream at a specified key. If the key does not exist, running this command will result in a key to be automatically created based on the entry.	XADD key ID field string [field string ...]
XCLAIM	Changes the ownership of a pending message, so that the new owner is the consumer specified as the command argument.	XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]
XDEL	Removes the specified entries from a stream, and returns the number of entries deleted, that may be different from the number of IDs passed to the command in case certain IDs do not exist.	XDEL key ID [ID ...]
XGROUP	Manages the consumer groups associated with a stream. You can use <b>XGROUP</b> to: <ul style="list-style-type: none"> <li>• Create a new consumer group associated with a stream.</li> <li>• Destroy a consumer group.</li> <li>• Remove a specified consumer from a consumer group.</li> <li>• Set the consumer group <i>last delivery ID</i> to something else.</li> </ul>	XGROUP [CREATE key groupname id-or-\$] [SETID key id-or-\$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]
XINFO	Retrieves different information about the streams and associated consumer groups.	XINFO [CONSUMERS key groupname] key key [HELP]
XLEN	Returns the number of entries in a stream. If the specified key does not exist, <b>0</b> is returned, indicating an empty stream.	XLEN key
XPENDING	Obtains data from a stream through a consumer group. This command is the interface to inspect the list of pending messages in order to observe and understand what clients are active, what messages are pending to be consumed, or to see if there are idle messages.	XPENDING key group [start end count] [consumer]

Command	Description	Syntax
XRANGE	Returns entries matching a given range of IDs.	XRANGE key start end [COUNT count]
XREAD	Reads data from one or multiple streams, only returning entries with an ID greater than the last received ID reported by the caller.	XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREADGROUP	A special version of the <b>XREAD</b> command, which is used to specify a consumer group to read from.	XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREVRANGE	This command is exactly like <b>XRANGE</b> , but with the notable difference of returning the entries in reverse order, and also taking the start-end range in reverse order.	XREVRANGE key end start [COUNT count]
XTRIM	Trims the stream to a specified number of items, if necessary, evicting old items (items with lower IDs).	XTRIM key MAXLEN [~] count

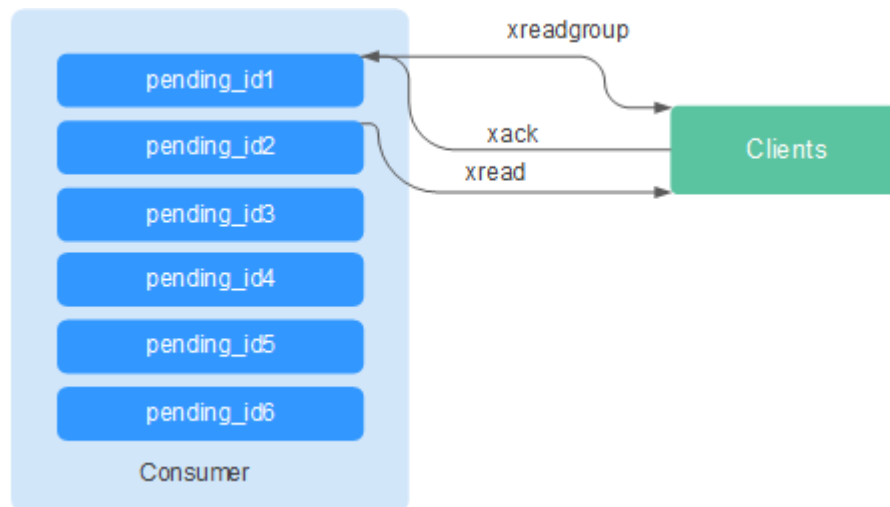
### Message (stream item) acknowledgement

Compared with Pub/Sub, streams not only support consumer groups, but also message acknowledgement.

When a consumer invokes the **XREADGROUP** command to read or invokes the **XCLAIM** command to take over a message, the server does not know whether the message is processed at least once. Therefore, once having successfully processed a message, the consumer should invoke the **XACK** command to notify the stream so that the message will not be processed again. In addition, the message is removed from PEL and the memory will be released from the Redis server.

In some cases, such as network faults, the client does not invoke **XACK** after consumption. In such cases, the item ID is retained in PEL. After the client is reconnected, set the start message ID of **XREADGROUP** to 0-0, indicating that all PEL messages and messages after **last\_id** are read. In addition, repeated message transmission must be supported when consumers consume messages.

Figure 5-4 Acknowledgment mechanism



## Memory Usage Optimization

The memory usage of Redis 5.0 is optimized based on the previous version.

- Active defragmentation

If a key is modified frequently and the value length changes constantly, Redis will allocate additional memory for the key. To achieve high performance, Redis uses the memory allocator to manage memory. Memory is not always freed up to the OS. As a result, memory fragments occur. If the fragmentation ratio (**used\_memory\_rss/used\_memory**) is greater than 1.5, the memory usage is inefficient.

To reduce memory fragments, properly plan and use cache data and standardize data writing.

For Redis 3.0 and earlier versions, memory fragmentation problems are resolved by restarting the process regularly. It is recommended that the actual cache data does not exceed 50% of the available memory.

For Redis 4.0, active defragmentation is supported, and memory is defragmented while online. In addition, Redis 4.0 supports manual memory defragmentation by running the **memory purge** command.

For Redis 5.0, improved active defragmentation is supported with the updated Jemalloc, which is faster, more intelligent, and provides lower latency.

- HyperLogLog implementation improvements

A HyperLogLog is a probabilistic data structure used to calculate the cardinality of a set while consuming little memory. Redis 5.0 improves HyperLogLog by further optimizing its memory usage.

For example: the B-tree is efficient in counting, but consumes a lot of memory. By using HyperLogLog, a lot of memory can be saved. While the B-tree requires 1 MB memory for counting, HyperLogLog needs only 1 KB.

- Enhanced memory statistics

The information returned by the **INFO** command is more detailed.

## New and Better Commands

### 1. Enhanced client management

- redis-cli supports cluster management.

In Redis 4.0 and earlier versions, the **redis-trib** module needs to be installed to manage clusters.

Redis 5.0 optimizes redis-cli, integrating all cluster management functions. You can run the **redis-cli --cluster help** command for more information.

- The client performance is enhanced in frequent connection and disconnection scenarios.

This optimization is valuable when your application needs to use short connections.

### 2. Simpler use of sorted sets

**ZPOPMIN** and **ZPOPMAX** commands are added for sorted sets.

- ZPOPMIN key [count]

Removes and returns up to **count** members with the lowest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with higher scores.

- ZPOPMAX key [count]

Removes and returns up to **count** members with the highest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with lower scores.

### 3. More sub-commands added to the help command

The **help** command can be used to view help information, saving you the trouble of visiting **redis.io** every time. For example, run the following command to view the stream help information: **xinfo help**

```
127.0.0.1:6379> xinfo help
1) XINFO <subcommand> arg arg ... arg. Subcommands are:
2) CONSUMERS <key> <groupname> -- Show consumer groups of group <groupname>.
3) GROUPS <key> -- Show the stream consumer groups.
4) STREAM <key> -- Show information about the stream.
5) HELP -- Print this help.
127.0.0.1:6379>
```

### 4. redis-cli command input tips

After you enter a complete command, redis-cli displays a parameter tip to help you memorize the syntax format of the command.

As shown in the following figure, run the **zadd** command, and redis-cli displays **zadd** syntax in light color.

```
# Cluster
cluster_enabled:0

# Keyspace
db0:keys=1,expires=0,avg_ttl=0
198.19.59.199:6379> zadd key [NX|XX] [CH] [INCR] score member [score member ...]
```

## RDB Storing LFU and LRU Information

In Redis 5.0, storage key eviction policies **LRU** and **LFU** were added to the RDB snapshot file.

- **FIFO**: First in, first out. The earliest stored data is evicted first.
- **LRU**: Least recently used. Data that is not used for a long time is evicted first.
- **LFU**: Least frequently used. Data that is least frequently used is evicted first.

### NOTE

The RDB file format of Redis 5.0 is modified and is backward compatible. Therefore, if a snapshot is used for migration, data can be migrated from the earlier Redis versions to Redis 5.0, but cannot be migrated from the Redis 5.0 to the earlier versions.

## 5.1.5 New Features of DCS for Redis 6.0

DCS for Redis 6.0 is compatible with the new features of the open-source Redis 6.0, in addition to all the improvements and commands in Redis 5.0.

### RESP3

Redis 6.0 introduced RESP3, the next-generation Redis protocol which implements more data types than RESP2.

- **Null**: a non-existent value. It replaces RESP2's **\*-1** and **-\$-1**.
- **Array**: an ordered collection
- **Simple string**: a space-efficient non-binary safe string
- **Blob string**: a binary safe string
- **Simple error**: a space-efficient non-binary safe error code or message
- **Blob error**: binary safe error code or message
- **Boolean**: true or false
- **Number**: a 64-bit signed integer
- **Big number**: a large number
- **Double**: a floating point number
- **Verbatim string**: a binary safe string with a text format
- **Map**: an unordered collection of key-value pairs
- **Set**: an unordered collection of non-repeated elements
- **Attribute**: attribute key-value pairs, similar to Map
- **Push**: out-of-band data, similar to Array, used by the Redis server to push data to the client.
- **Hello**: response returned by the **HELLO** command, sent when the connection between the client and the server is established.

### NOTE

To use RESP3, ensure that the client SDK supports it. Otherwise, the client SDK communicates with the server through **HELLO** of RESP2.

## Client-side Caching

Redis 6.0 uses tracking to proactively instruct clients to refresh their cache. There are three implementation modes:

### RESP3

- Default mode
- Broadcasting mode

### RESP2

- Redirecting mode

Format for enabling client-side cache tracking:

```
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix] [BCAST] [OPTIN][OPTOUT] [NOLOOP]
```

In RESP3, the server uses Push messages to send notifications. In the default mode, Redis remembers the keys requested by each client. When the value of a key changes, Redis sends an invalidation message to notify the corresponding client. Note that Redis notifies each client only once. If the value changes again later, the client must read the key again to enable tracking. To enable tracking in the default mode, use the following command:

```
CLIENT TRACKING ON
```

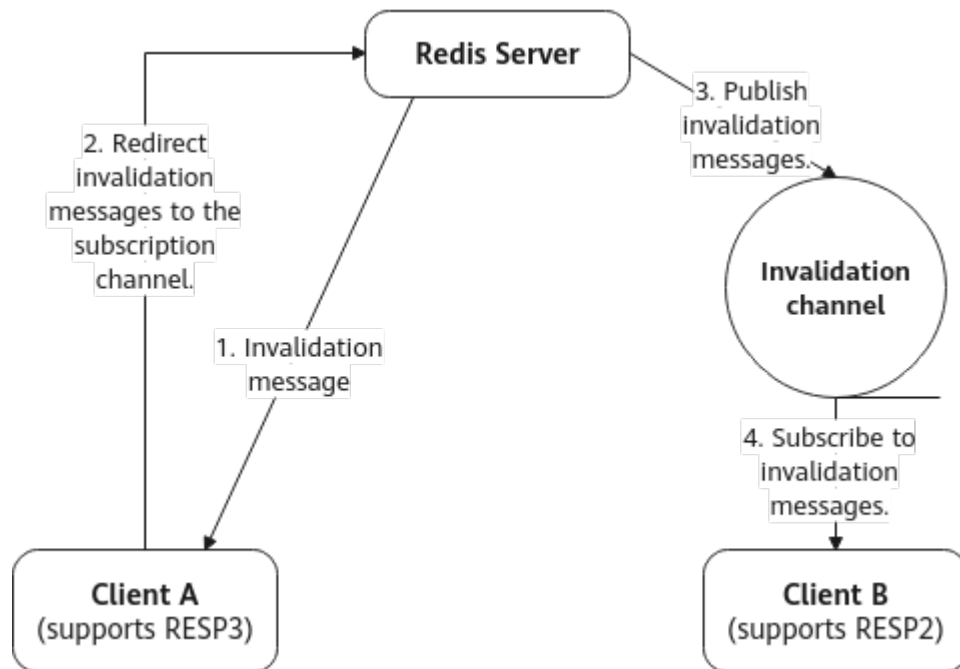
In the broadcasting mode, all clients are notified when the value of the key matching a tracked key prefix changes. If there is large number of keys that match the key prefix or there is large number of changes, the server sends many invalidation messages, consuming much network bandwidth. To enable tracking in the broadcasting mode, use the following command:

```
CLIENT TRACKING ON BCAST PREFIX key-prefix
```

If the client SDK does not support RESP3, only the redirecting mode of RESP2 can be used to implement tracking. In this case, a dedicated client that supports RESP3 needs to act as the transit node that redirects invalidation messages from Redis to specific subscription channels.



Figure 5-5 Client cache tracking (redirecting mode)



## Faster RDB Loading

In Redis 6.0, the loading of RDB files is optimized for a 20% to 30% increase in speed.

## Optimized INFO Command

The processing of the **INFO** command is optimized, especially in the scenario where a large number of clients are connected, resulting in higher performance and lower latency.

## 5.2 Client and Network Connection

### 5.2.1 Security Group Configurations

DCS helps you control access to your DCS instances in the following ways, depending on the deployment mode:

- To control access to DCS Redis 3.0 instances, you can use security groups. Whitelists are not supported. Security group operations are described in this section.
- To control access to DCS Redis 4.0/5.0/6.0 instances, you can use whitelists. Security groups are not supported. Whitelist operations are described in [Managing IP Address Whitelist](#).

The following describes how to configure security groups for **intra-VPC access** to DCS Redis 3.0 instances.

## Intra-VPC Access to DCS Redis 3.0 Instances

An ECS can communicate with a DCS instance if they belong to the same VPC and security group rules are configured correctly.

In addition, you must configure correct rules for the security groups of both the ECS and DCS instance so that you can access the instance through your client.

- If the ECS and DCS instance are configured with the same security group, network access in the group is not restricted by default.
- If the ECS and DCS instance are configured with different security groups, add security group rules to ensure that the ECS and DCS instance can access each other.

### NOTE

- Suppose that the ECS on which the client runs belongs to security group **sg-ECS**, and the DCS instance that the client will access belongs to security group **sg-DCS**.
  - Suppose that the port number of the DCS service is 6379.
  - The remote end is a security group or an IP address.
- a. Configuring security group for the ECS.  
Add the following outbound rule to allow the ECS to access the DCS instance. Skip this rule if there are no restrictions on the outbound traffic.
  - b. Configuring security group for the DCS instance.  
To ensure that your client can access the DCS instance, add the following inbound rule to the security group configured for the DCS instance:

---

### NOTICE

In the inbound rule of the security group configured for the DCS instance, set the remote end to an IP address in the same CIDR block as the subnet.

To prevent ECSs bound with same security group as the DCS instance from being attacked by Redis vulnerabilities, exercise caution when using **0.0.0.0/0**.

---

## 5.2.2 Does DCS Support Access at EIPs?

No. DCS instances cannot be access at their EIPs. To ensure security, the ECS that serves as a client and the DCS instance that the client will access must belong to the same VPC.

## 5.2.3 Does DCS Support Cross-VPC Access?

Cross-VPC means the client and the instance are not in the same VPC.

Generally, VPCs are isolated from each other and ECSs cannot access DCS instances that belong to a different VPC from these ECSs.

However, by establishing VPC peering connections between VPCs, ECSs can access single-node and master/standby DCS instances across VPCs.

When using VPC peering connections to access DCS instances across VPCs, adhere to the following rules:

- If network segments 172.16.0.0/12 to 172.16.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24.
- If network segments 192.168.0.0/16 to 192.168.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.
- If network segments 10.0.0.0/8 to 10.0.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.

For more information about VPC peering connection, see "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.

#### NOTICE

Cluster DCS Redis instances do not support cross-VPC access. ECSs in a VPC cannot access cluster DCS instances in another VPC by using VPC peering connections.

## 5.2.4 What Should I Do If Access to DCS Fails After Server Disconnects?

**Analysis:** If persistent connections ("pconnect" in Redis terminology) or connection pooling is used and connections are closed after being used for connecting to DCS instances, errors will be returned at attempts to reuse the connections.

**Solution:** When using pconnect or connection pooling, do not close the connection after the end of a request. If the connection is dropped, re-establish it.

## 5.2.5 Why Do Requests Sometimes Time Out in Clients?

Occasional timeout errors are normal because of network connectivity and client timeout configurations.

You are advised to include reconnection operations into your service code to avoid service failure if a single request fails.

If a connection request times out, check if AOF persistence has been enabled. To avoid blocking, ensure that AOF has been enabled.

If timeout errors occur frequently, contact O&M personnel.

## 5.2.6 What Should I Do If an Error Is Returned When I Use the Jedis Connection Pool?

The error message that will possibly be displayed when you use the Jedis connection pool is as follows:

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
```

If this error message is displayed, check whether your instance is running properly. If it is running properly, perform the following checks:

### Step 1 Network

1. Check the IP address configurations.

Check whether the IP address configured on the Jedis client is the same as the subnet address configured for your DCS instance.

2. Test the network.

Use the ping command and telnet on the client to test the network.

- If the network cannot be pinged:

For intra-VPC access to a DCS Redis 3.0 instance, ensure that the client and your DCS instance belong to the same VPC and security group, or the security group of your DCS instance allows access through port 6379. For details, see [Security Group Configurations](#).

- If the IP address can be pinged but telnet failed, restart your instance. If the problem persists after the restart, contact technical support.

### Step 2 Check the number of connections.

Check whether the number of established network connections exceeds the upper limit configured for the Jedis connection pool. If the number of established connections approaches the configured upper limit, restart the DCS service and check whether the problem persists. If the number of established connections is far below the upper limit, continue with the following checks.

In Unix or Linux, run the following command to query the number of established network connections:

```
netstat -an | grep 6379 | grep ESTABLISHED | wc -l
```

In Windows, run the following command to query the number of established network connections:

```
netstat -an | find "6379" | find "ESTABLISHED" /C
```

### Step 3 Check the JedisPool code.

If the number of established connections approaches the upper limit, determine whether the problem is caused by service concurrency or incorrect usage of JedisPool.

When using JedisPool, you must call `jedisPool.returnResource()` or `jedis.close()` (recommended) to release the resources after you call `jedisPool.getResource()`.

### Step 4 Check the number of TIME\_WAIT connections.

Run the `ss -s` command to check whether there are too many `TIME_WAIT` connections on the client.

```
root@heru-nodelete:~# ss -s
Total: 140 (kernel 240)
TCP: 11 (estab 3, closed 1, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total IP IPv6
* 240 - -
RAW 0 0 0
UDP 2 2 0
TCP 10 6 4
INET 12 8 4
FRAG 0 0 0
```

If there are too many **TIME\_WAIT** connections, modify the kernel parameters by running the **/etc/sysctl.conf** command as follows:

```
##Uses cookies to prevent some SYN flood attacks when the SYN waiting queue overflows.  
net.ipv4.tcp_syncookies = 1  
##Reuses TIME_WAIT sockets for new TCP connections.  
net.ipv4.tcp_tw_reuse = 1  
##Enables quick reclamation of TIME_WAIT sockets in TCP connections.  
net.ipv4.tcp_tw_recycle = 1  
##Modifies the default timeout time of the system.  
net.ipv4.tcp_fin_timeout = 30
```

After the modification, run the **/sbin/sysctl -p** command for the modification to take effect.

**Step 5** If the problem persists after you perform the preceding checks, perform the following steps.

Capture packets and send packet files along with the time and description of the exception to technical support for analysis.

Run the following command to capture packets:

```
tcpdump -i eth0 tcp and port 6379 -n -nn -s 74 -w dump.pcap
```

In Windows, you can also install the Wireshark tool to capture packets.

 **NOTE**

Replace the NIC name to the actual one.

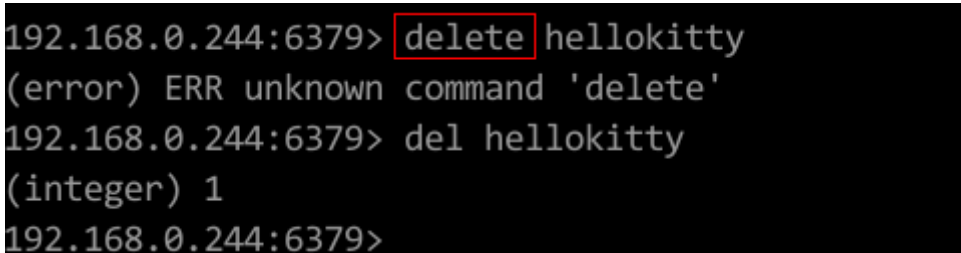
----End

## 5.2.7 Why Is "ERR unknown command" Displayed When I Access a DCS Redis Instance Through a Redis Client?

The possible causes are as follows:

1. The command is spelled incorrectly.

As shown in the following figure, the error message is returned because the correct command for deleting a string should be **del**.



```
192.168.0.244:6379> delete hellokitty  
(error) ERR unknown command 'delete'  
192.168.0.244:6379> del hellokitty  
(integer) 1  
192.168.0.244:6379>
```

2. A command available in a higher Redis version is run in a lower Redis version.

As shown in the following figure, the error message is returned because a stream command (available in Redis 5.0) is run in Redis 3.0.

```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

3. Some commands are disabled.  
DCS Redis instance interfaces are fully compatible with the open-source Redis in terms of data access. However, for ease of use and security purposes, some operations cannot be initiated through Redis clients. For details about disabled commands, see [Command Compatibility](#).

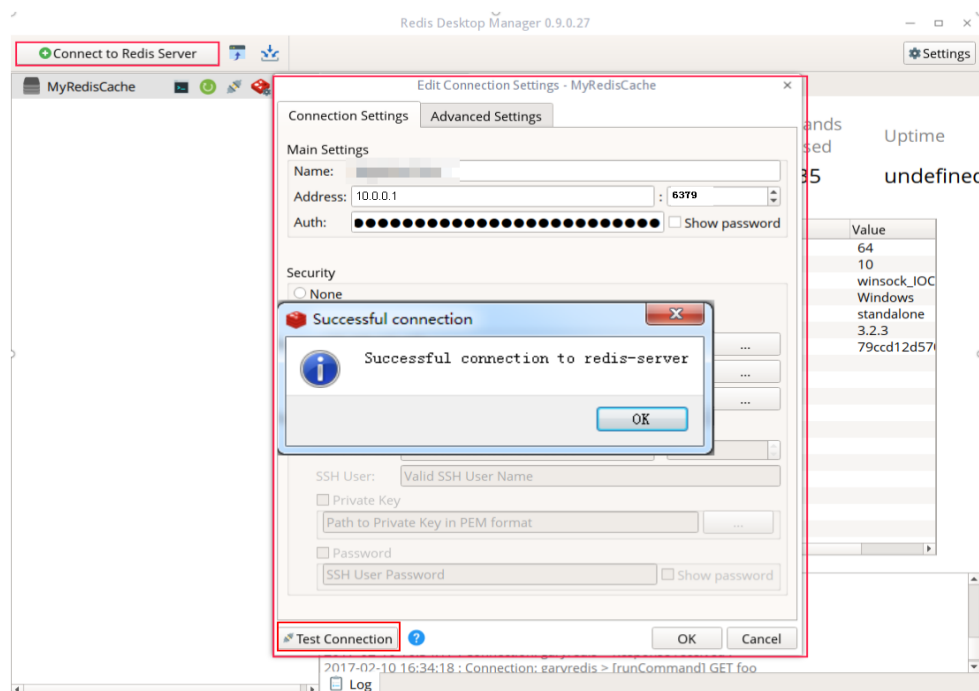
## 5.2.8 How Do I Access a DCS Redis Instance Through Redis Desktop Manager?

You can access a DCS Redis instance through the Redis Desktop Manager within a VPC.

1. Enter the address, port number (6379), and authentication password of the DCS instance you want to access.
2. Click **Test Connection**.

The system displays a success message if the connection is successful.

**Figure 5-6** Accessing a DCS Redis instance through Redis Desktop Manager over the intranet



 NOTE

When accessing a cluster DCS instance, the Redis command is run properly, but an error message may display on the left because DCS clusters are based on Codis, which differs from the native Redis in terms of the **INFO** command output.

## 5.2.9 What If "ERR Unsupported CONFIG subcommand" is Displayed in SpringCloud?

By using DCS Redis instances, Spring Session can implement session sharing. When interconnecting with Spring Cloud, the following error information is displayed:

Figure 5-7 Spring Cloud error information

```
...redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource - {dataSource-2} closed
2019-02-01 00:36:59 ERROR org.springframework.web.context.ContextLoader - Context initialization failed
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'enableRedisKeyspaceNotificationsInitializer' defined in class path resource [org/springframework/session/data/redis/config/annotation/web/http/RedisHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessUsageException: ERR Unsupported CONFIG command; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1794)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:583)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:562)
    at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:313)
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:228)
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:318)
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:298)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:755)
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:868)
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:449)
```

For security purposes, DCS does not support the **CONFIG** command initiated by a client. You need to perform the following steps:

1. On the DCS console, set the value of the **notify-keyspace-event** parameter to **Egx** for a DCS Redis instance.
2. Add the following content to the XML configuration file of the Spring framework:

```
<util:constant
static-
field="org.springframework.session.data.redis.config.ConfigureRedisAction.NO_OP"/>
```

3. Modify the related Spring code. Enable the **ConfigureRedisAction.NO\_OP** bean component to forbid a client to invoke the **CONFIG** command.

```
@Bean
public static ConfigureRedisAction configureRedisAction() {
    return ConfigureRedisAction.NO_OP;
}
```

For more information, see the [Spring Session Documentation](#).

**NOTICE**

Session sharing is supported by single-node and master/standby DCS Redis instances, but not by cluster DCS Redis instances.

## 5.2.10 How Do I Troubleshoot Redis Connection Failures?

Preliminary checks:

- Check the connection address.

Obtain the connection address from the instance basic information page on the DCS console.

- Check the instance password.  
If the instance password is incorrect, the port can still be accessed but the authentication will fail.
- Check the port.  
Port 6379 is the default port used in intra-VPC access to a DCS Redis instance.
- Check if the maximum bandwidth has been reached.  
If the bandwidth reaches the maximum bandwidth for the corresponding instance specifications, Redis connections may time out.
- For a DCS Redis 3.0 instance, check the inbound access rules of the security group.  
Intra-VPC access: If the Redis client and the Redis instance are bound with different security groups, allow inbound access over port 6379 for the security group of the instance.  
For details, see [Security Group Configurations](#).
- For a DCS Redis 4.0/5.0 instance, check the whitelist configuration.  
If the instance has a whitelist, ensure that the client IP address is included in the whitelist. Otherwise, the connection will fail.  
For details, see [Managing IP Address Whitelist](#).  
If the client IP address has changed, add the new IP address to the whitelist.
- Check the configuration parameter **notify-keyspace-events**.  
Set **notify-keyspace-events** to **Egx**.

**Further checks:**

- Jedis connection pool error
- Error "Read timed out" or "Could not get a resource from the pool"  
Check if the **KEYS** command has been used. This command consumes a lot of resources and can easily block Redis. Instead, use the **SCAN** command and avoid executing the command frequently.

## 5.2.11 What Should Be Noted When Using Redis for Pub/Sub?

Pay attention to the following issues when using Redis for pub/sub:

- Your client must process messages in a timely manner.  
Your client subscribes to a channel. If it does not receive messages in a timely manner, DCS instance messages may be overstocked. If the size of accumulated messages reaches the threshold (32 MB by default) or remains at a certain level (8 MB by default) for a certain period of time (1 minute by default), your client will be automatically disconnected to prevent server memory exhaustion.
- Your client must support connection re-establishment in case of disconnection.  
In the event of a disconnection, you need to run the **subscribe** or **psubscribe** command on your client to subscribe to a channel again. Otherwise, your client cannot receive messages.



- Do not use pub/sub in scenarios with high message reliability requirements. The Redis pub/sub is not a reliable messaging system. Messages that are not retrieved will be discarded when your client is disconnected or a master/standby switchover occurs.

## 5.2.12 Should I Use a Domain Name or an IP Address to Connect to a DCS Redis Instance?

- Single-node, and Proxy Cluster:  
Each instance has only one IP address and one domain name address. The addresses remain unchanged before and after master/standby switchover. You can use either address to connect to the instance.
- Master/Standby:  
Each instance has one IP address and two domain name addresses. One of the domain name addresses is used only for processing read requests. The addresses remain unchanged after master/standby switchover. You can use any address to connect to the instance.  
When you use a domain name address, distinguish between read and write requests. If you use **Connection Address** or **IP Address**, functions are not affected. If you use **Read-only Address**, only read requests are processed.
- Redis Cluster:  
A Redis Cluster instance has multiple pairs of master and replica IP addresses and one domain name address. You can use any address to connect to the instance.  
The connected node sends requests to the correct node. All nodes in the cluster can receive requests. **Configure multiple or all IP addresses** to prevent single points of failure.

### NOTE

- For details about how to connect to an instance, see the *Developer Guide*.

## 5.3 Redis Usage

### 5.3.1 Why Is CPU Usage of a DCS Redis Instance 100%?

- Possible cause 1:  
The service QPS is so high that the CPU usage spikes to 100%.
- Possible cause 2:  
You have run commands that consume a lot of resources, such as **KEYS**. This will make CPU usage spike and can easily trigger a master/standby switchover.

### 5.3.2 Can I Change the VPC and Subnet for a DCS Redis Instance?

No. Once an instance is created, its VPC and subnet cannot be changed. If you want to use a different set of VPC and subnet, create a same instance and specify

a desired set of VPC and subnet. After the new instance is created, you can migrate data from the old instance to the new instance by following the [data migration instructions](#).

### 5.3.3 Why Aren't Security Groups Configured for DCS Redis 4.0 and Later Instances?

Currently, DCS Redis 4.0 and later instances use VPC endpoints and do not support security groups.

### 5.3.4 Do DCS Redis Instances Limit the Size of a Key or Value?

- The maximum allowed size of a key is 512 MB.  
To reduce memory usage and facilitate key query, ensure that each key does not exceed 1 KB.
- The maximum allowed size of a string is 512 MB.
- The maximum allowed size of a Set, List, or Hash is 512 MB.  
In essence, a Set is a collection of Strings; a List is a list of Strings; a Hash contains mappings between string fields and string values.

Prevent the client from constantly writing large values in Redis. Otherwise, network transmission efficiency will be lowered and the Redis server would take a longer time to process commands, resulting in higher latency.

### 5.3.5 Can I Obtain the Addresses of the Nodes in a Cluster DCS Redis Instance?

Cluster DCS Redis 3.0 instances (Proxy Cluster type) are used in the same way that you use single-node or master/standby instances. You do not need to know the backend node addresses.

For a cluster DCS Redis 4.0 or later instance (Redis Cluster type), run the **CLUSTER NODES** command to obtain node addresses:

```
redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes
```

In the output similar to the following, obtain the IP addresses and port numbers of all the master nodes.

```
[root@ecs-54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a 123 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-1040d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413f0
be6c07faa64d724323e0d7cedc3f38346dcdbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985ff
c16b9acaeeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb79
```

### 5.3.6 Why Is Available Memory Smaller Than Instance Cache Size?

DCS Redis 3.0 instances are deployed on VMs and some memory is reserved for system overheads. This problem will not occur on other instance versions.

### 5.3.7 Does DCS for Redis Support Multiple Databases?

Both single-node and master/standby DCS Redis instances support multiple databases. By default, single-node and master/standby DCS instances can read and write data in 256 databases (databases numbering 0–255). The default database is DB0.

Redis Cluster DCS instances do not support multi-DB. There is only one database (DB0).

### 5.3.8 Does DCS for Redis Support Redis Clusters?

Yes. DCS for Redis 4.0 and later support Redis Clusters. DCS for Redis 3.0 supports Proxy Clusters.

### 5.3.9 Does DCS for Redis Support Sentinel?

Cluster instances and master/standby DCS Redis 4.0 and later instances support Sentinels. Sentinels monitor the running status of both the master and standby nodes of a master/standby instance and each shard of a cluster instance. If the master node becomes faulty, a failover will be performed.

DCS for Redis 3.0 does not support Redis Sentinel. Instead, it uses keepalive to monitor master and replica nodes and to manage failovers.

### 5.3.10 What Is the Default Data Eviction Policy?

Data is evicted from cache based on a user-defined space limit in order to make space for new data. For details, see the [Redis official website](#). You can [view or change the eviction policy](#) by configuring an instance parameter on the DCS console.

#### Eviction Policies Supported by DCS Redis Instances

When **maxmemory** is reached, you can select one of the following eight eviction policies:

- **noeviction**: When the memory limit is reached, DCS instances return errors to clients and no longer process write requests and other requests that could result in more memory to be used. However, **DEL** and a few more exception requests can continue to be processed.
- **allkeys-lru**: DCS instances try to evict the least recently used keys first, in order to make space for new data.
- **volatile-lru**: DCS instances try to evict the least recently used keys with an expire set first, in order to make space for new data.
- **allkeys-random**: DCS instances recycle random keys so that new data can be stored.
- **volatile-random**: DCS instances evict random keys with an expire set, in order to make space for new data.
- **volatile-ttl**: DCS instances evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first, in order to make space for new data.
- **allkeys-lfu**: DCS instances evict the least frequently used keys from all keys.

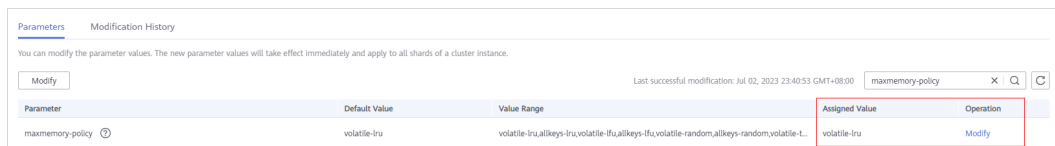
- **volatile-lfu**: DCS instances evict the least frequently used keys with an **expire** field from all keys.

 **NOTE**

If no key can be recycled, **volatile-lru**, **volatile-random**, and **volatile-ttl** are the same as **noeviction**. For details, see the description of **noeviction**.

## Viewing or Changing Eviction Policies

You can view or change the eviction policy with the **maxmemory-policy** parameter.



The screenshot shows a web interface for configuring parameters. At the top, there are tabs for 'Parameters' and 'Modification History'. Below the tabs, a message states: 'You can modify the parameter values. The new parameter values will take effect immediately and apply to all shards of a cluster instance.' There is a 'Modify' button and a search bar containing 'maxmemory-policy'. A table lists the parameter details:

Parameter	Default Value	Value Range	Assigned Value	Operation
maxmemory-policy	volatile-lru	volatile-lru,allkeys-lru,volatile-lfu,allkeys-lfu,volatile-random,allkeys-random,volatile-L	volatile-lru	Modify

### 5.3.11 What Should I Do If an Error Occurs in Redis Exporter?

Start the Redis exporter using the CLI. Based on the output, check for errors and troubleshoot accordingly.

```
[root@ecs-swk /]# ./redis_exporter -redis.addr 192.168.0.23:6379
INFO[0000] Redis Metrics Exporter V0.15.0 build date:2018-01-19-04:08:01 sha1:
a0d9ec4704b4d35cd08544d395038f417716a03a
Go:go1.9.2
INFO[0000] Providing metrics at :9121/metrics
INFO[0000] Connecting to redis hosts: []string{192.168.0.23:6379}
INFO[0000] Using alias:[]string{""}
```

### 5.3.12 Why Is Redisson Distributed Lock Not Supported by DCS Proxy Cluster Redis 3.0 Instances?

Redisson implements lock acquisition and unlocking in the following process:

1. Redisson lock acquisition and unlocking are implemented by running Lua scripts.
2. During lock acquisition, the **EXISTS**, **HSET**, **PEXPIRE**, **HEXISTS**, **HINCRBY**, **PEXPIRE**, and **PTTL** commands must be executed in the Lua script.
3. During unlocking, the **EXISTS**, **PUBLISH**, **HEXISTS**, **PEXPIRE**, and **DEL** commands must be executed in the Lua script.

In a proxy-based cluster, the proxy processes **PUBLISH** and **SUBSCRIBE** commands and forwards requests to the Redis server. The **PUBLISH** command cannot be executed in the Lua script.

As a result, Proxy Cluster DCS Redis 3.0 instances do not support Redisson distributed locks. To use Redisson, resort to Redis 4.0 or 5.0 instead.

### 5.3.13 Can I Customize or Change the Port for Accessing a DCS Instance?

You cannot customize or change the port for accessing a DCS Redis 3.0 instance. You can customize (during instance creation) and change the port (on the instance details page) for accessing a DCS Redis 4.0 or later instance.

- Redis 3.0  
Intra-VPC access: port 6379.
- Redis 4.0 and later  
You can specify a port (ranging from 1 to 65535) or use the default port (6379) for accessing an instance. If no port is specified, the default port will be used.

If the instance and the client use different security groups, you must configure access rules for the security groups, allowing access through the specified port. For details, see [Security Group Configurations](#).

### 5.3.14 Can I Modify the Connection Addresses for Accessing a DCS Instance?

After a DCS instance is created, its intra-VPC connection addresses cannot be modified.

To use a different IP address, you must create a new instance and manually specify an IP address. After the instance is created, migrate the data from the old instance to the new instance.

For details about accessing DCS instances through clients, see *Distributed Cache Service Developer Guide* > "Accessing an Instance".

### 5.3.15 Does DCS Support Cross-AZ Deployment?

Master/Standby and cluster DCS Redis instances can be deployed across availability zones (AZs).

- If instance nodes in an AZ are faulty, nodes in other AZs will not be affected. The standby node automatically becomes the master node to continue to operate, ensuring disaster recovery (DR).
- Cross-AZ deployment does not compromise the speed of data synchronization between the master and standby nodes.

### 5.3.16 Why Does It Take a Long Time to Start a Cluster DCS Instance?

Possible cause: When a cluster instance is started, status and data are synchronized between the nodes of the instance. If a large amount of data is continuously written into the instance before the synchronization is complete, the synchronization will be prolonged and the instance remains in the **Starting** state. After the synchronization is complete, the instance enters the **Running** state.

Solution: Start writing data to an instance only after the instance has been started.

### 5.3.17 What If Redis Commands Are Incompatible with DCS for Redis?

You can verify whether the commands used by your applications are compatible with DCS by analyzing your service commands.

1. DCS for Redis 3.0 has integrated Redis 4.0 commands, which can be used properly in most cases. However, some commands are not supported or are disabled. For details about unsupported commands, see [Command Compatibility](#).
2. DCS for Redis 3.0 does not support Redis 5.0 commands, such as Stream commands.

### 5.3.18 Does DCS for Redis Provide Backend Management Software?

No. If you wish to query Redis configurations and usage information, use `redis-cli`. If you wish to monitor DCS Redis instance metrics, go to the Cloud Eye console. For details on how to configure and view the metrics, see [Monitoring](#).

### 5.3.19 Why Is Memory of a DCS Redis Instance Used Up by Just a Few Keys?

Possible cause: The output buffer may have occupied an excessive amount of memory.

Solution: After connecting to the instance using `redis-cli`, run the `redis-cli --bigkeys` command to scan for big keys. Then, run the `info` command to check the output buffer size.

### 5.3.20 Can I Recover Data from Deleted DCS Instances?

If a DCS instance is automatically deleted or manually deleted through the Redis client, its data cannot be retrieved. If you have backed up the instance, you can restore its data from the backup. However, the restoration will overwrite the data written in during the period from the backup and the restoration.

You can restore backup data to a master/standby cluster, or instance through **Backups & Restorations** on the DCS console. For details, see [Restoring a DCS Instance](#).

If a DCS instance is deleted, the instance data and its backup will also be deleted. Before deleting an instance, you can download the backup files of the instance for permanent local storage and can also migrate them to a new instance if you need to restore the data. For details about how to download the backup data, see [Downloading a Backup File](#)

### 5.3.21 Why Is "Error in execution" Returned When I Access Redis?

**Symptom:** "Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'" is returned during a Redis connection.

**Analysis:** An out-of-memory (OOM) error indicates that the maximum memory is exceeded. In the error information, the "maxmemory" parameter indicates the maximum memory configured on the Redis server.

If the memory usage of the Redis instance is less than 100%, the memory of the node where data is written may have reached the maximum limit. Connect to

each node in the cluster by running **redis-cli -h <redis\_ip> -p 6379 -a <redis\_password> -c --bigkeys**. When connecting to a replica node, run the **READONLY** command before running the **bigkeys** command.

## 5.4 Redis Commands

### 5.4.1 How Do I Clear Redis Data?

Exercise caution when clearing data.

- Redis 3.0  
Data of a DCS Redis 3.0 instance cannot be cleared on the console, and can only be cleared by the **FLUSHDB** or **FLUSHALL** command in redis-cli.  
Run the **FLUSHALL** command to clear all the data in the instance.  
Run the **FLUSHDB** command to clear the data in the currently selected DB.
- Redis 4.0 and later  
To clear data of a DCS Redis 4.0 and later instance, you can run the **FLUSHDB** or **FLUSHALL** command in redis-cli, use the data clearing function on the DCS console, or run the **FLUSHDB** command on Web CLI.  
To clear data of a Redis Cluster instance, run the **FLUSHDB** or **FLUSHALL** command on every shard of the instance. Otherwise, data may not be completely cleared.

#### NOTE

- Currently, only DCS Redis 4.0 and later instances support data clearing by using the DCS console and by running the **FLUSHDB** command on Web CLI.
- When you run the **FLUSHDB** command on Web CLI, only one shard is cleared at a time. If there are multiple shards, connect to the master node of each shard and run the **FLUSHDB** command separately.
- Redis Cluster data cannot be cleared by using Web CLI.

### 5.4.2 How Do I Disable High-Risk Commands?

After creating a DCS Redis 4.0 or later instance, you can rename the following critical commands: Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL** commands.

Rename them during instance creation or on the console after the instance is created. To do so, choose **More > Command Renaming** in the instance list.

#### NOTE

- Currently, Redis does not support disabling of commands. To avoid risks when using the preceding commands, rename them. For details about the supported and disabled commands in DCS, see [Command Compatibility](#).
- The system will restart the instance after you rename commands. The new commands take effect after the restart.
- Remember the new command names because they will not be displayed on the console for security purposes.

### 5.4.3 Does DCS for Redis Support Pipelining?

Yes.

For Redis Cluster instances, ensure that all commands in a pipeline are executed on the same shard.

### 5.4.4 Does DCS for Redis Support the INCR and EXPIRE Commands?

Yes. For more information about Redis command compatibility, see [Command Compatibility](#).

### 5.4.5 Why Do I Fail to Execute Some Redis Commands?

Possible causes include the following:

- The command is incorrect.

As shown in the following figure, the error message is returned because the correct command for deleting a key should be **del**.

```
192.168.0.244:6379> delete hellokitty
(error) ERR unknown command 'delete'
192.168.0.244:6379> del hellokitty
(integer) 1
192.168.0.244:6379>
```

- A command available in a higher Redis version is run in a lower Redis version.

As shown in the following figure, the error message is returned because a stream command (available in Redis 5.0) is run in Redis 3.0.

```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

- The command is disabled in DCS.

For security purposes, some Redis commands are disabled in DCS. For details about disabled and restricted Redis commands, see [Command Compatibility](#).

- The LUA script fails to be executed.

For example, the error message "ERR unknown command 'EVAL'" indicates that your DCS Redis instance is of a lower version that does not support the LUA script. In this case, contact technical support for the instance to be upgraded.

- The **CLIENT SETNAME** and **CLIENT GETNAME** commands fail to be executed.



This is because the DCS Redis instance is of a lower version that does not support these commands. In this case, contact technical support for the instance to be upgraded.

## 5.4.6 Why Does a Redis Command Fail to Take Effect?

Run the command in redis-cli to check whether the command takes effect.

The following describes two scenarios:

- Scenario 1: Set and query the value of a key to check whether the **SET** and **GET** commands work.

The **SET** command is used to set the string value. If the value is not changed, run the following commands in redis-cli to access the instance:

```
192.168.2.2:6379> set key_name key_value
OK
192.168.2.2:6379> get key_name
"key_value"
192.168.2.2:6379>
```

- Scenario 2: If the timeout set using the **EXPIRE** command is incorrect, perform the following operations:

Set the timeout to 10 seconds and run the **TTL** command to view the remaining time. As shown in the following example, the remaining time is 7 seconds.

```
192.168.2.2:6379> expire key_name 10
(integer) 1
192.168.2.2:6379> ttl key_name
(integer) 7
192.168.2.2:6379>
```

### NOTE

Redis clients (including redis-cli, Jedis clients, and Python clients) communicate with Redis server using a binary protocol.

If Redis commands are run properly in redis-cli, the problem may lie in the service code. In this case, create logs in the code for further analysis.

## 5.4.7 Is There a Time Limit on Executing Redis Commands? What Will Happen If a Command Times Out?

Redis timeouts happen on the client or server.

- Timeouts on the client are managed in the client code. Services can determine an appropriate timeout. For example, parameter **timeout** can be configured on Java Lettuce.

When a command times out on the client, errors, command blocks, or client connection retries may occur.

- On the server, the **timeout** parameter is set to **0** by default, indicating that connections will not be terminated. To modify the setting, see [Modifying Configuration Parameters](#).

If the **timeout** parameter is not **0**, idle connections between the client and server will be terminated as specified.

## 5.5 Instance Scaling and Upgrade

### 5.5.1 Can DCS Redis Instances Be Upgraded, for Example, from Redis 4.0 to 5.0?

No. Different Redis versions use different underlying architectures. The Redis version used by a DCS instance cannot be changed once the instance is created. For example, you cannot change a DCS Redis 4.0 instance to Redis 5.0. However, you will be informed of any defects or problems found in Redis.

If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the original instance to the new one. For details on how to migrate data, see [Migrating Data with DCS](#).

### 5.5.2 Are Services Interrupted If Maintenance is Performed During the Maintenance Time Window?

O&M personnel will contact you before performing maintenance during the maintenance time window, informing you of the operations and their impacts. You do not need to worry about instance running exceptions.

### 5.5.3 Are Instances Stopped or Restarted During Specification Modification?

No. Specification modifications can take place while the instance is running and do not affect any other resources.

### 5.5.4 Are Services Interrupted During Specification Modification?

You are advised to change the instance specifications during off-peak hours because specification modification has the following impacts:

#### Change of the Instance Type

- **Supported instance type changes:**
  - From single-node to master/standby: Supported by Redis 3.0, and not by Redis 4.0/5.0.
  - From master/standby to Proxy Cluster: Supported by Redis 3.0, and not by Redis 4.0/5.0/6.0.  
If the data of a master/standby DCS Redis 3.0 instance is stored in multiple databases, or in non-DB0 databases, the instance cannot be changed to the Proxy Cluster type. A master/standby instance can be changed to the Proxy Cluster type only if its data is stored only on DB0.
  - From cluster types to other types: Not supported.
- **Impact of instance type changes:**
  - From single-node to master/standby for a DCS Redis 3.0 instance:

- The instance cannot be connected for several seconds and remains read-only for about 1 minute.
- From master/standby to Proxy Cluster for a DCS Redis 3.0 instance:  
The instance cannot be connected and remains read-only for 5 to 30 minutes.

## Scaling

- The following table lists scaling options supported by different DCS instances.

**Table 5-5** Scaling options supported by different DCS instances

Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster
Redis 3.0	Scaling up/down	Scaling up/down	N/A	Scaling out
Redis 4.0/5.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/down, out/in, and replica quantity change	Scaling up/down and out/in
Redis 6.0	Scaling up/down	Scaling up/down	Scaling up/down, out/in, and replica quantity change	N/A

### NOTE

If the reserved memory of a DCS Redis 3.0 instance is insufficient, the scaling may fail when the memory is used up.

Change the replica quantity and capacity separately.

- **Impact of scaling:**
  - Single-node and master/standby
    - A DCS Redis 4.0 or later instance will be disconnected for several seconds and remain read-only for about 1 minute.
    - A DCS Redis 3.0 instance will be disconnected and remain read-only for 5 to 30 minutes.
    - For scaling up, only the memory of the instance is expanded. The CPU processing capability is not improved.
    - Data of single-node instances may be lost because they do not support data persistence. After scaling, check whether the data is complete and import data if required.

- Backup records of master/standby instances cannot be used after scaling down.
- Cluster
  - If the shard quantity is not decreased, the instance can always be connected, but the CPU usage will increase, compromising performance by up to 20%, and the latency will increase during data migration.
  - During scaling up, new Redis Server nodes are added, and data is automatically balanced to the new nodes.
  - Nodes will be deleted if the shard quantity decreases. To prevent disconnection, ensure that the deleted nodes are not directly referenced in your application.
  - Ensure that the used memory of each node is less than 70% of the maximum memory per node of the new flavor. Otherwise, you cannot perform the scale-in.
  - If the memory becomes full during scaling due to a large amount of data being written, scaling will fail. Modify specifications during off-peak hours.
  - Scaling involves data migration. The latency for accessing the key being migrated increases. For a Redis Cluster instance, ensure that the client can properly process the **MOVED** and **ASK** commands. Otherwise, requests will fail.
  - Backup records created before scaling cannot be restored.
- **Notes on changing the number of replicas of a DCS Redis instance:**  
Deleting replicas interrupts connections. If your application cannot reconnect to Redis or handle exceptions, you need to restart the application after scaling.

### 5.5.5 Why Can't I Modify Specifications for a DCS Redis Instance?

- Check whether other tasks are running.  
Specifications of a DCS instance cannot be modified if another task of the instance is still running. For example, you cannot delete or scale up an instance while it is being restarted. Likewise, you cannot delete an instance while it is being scaled up.  
If the specification modification fails, try again later. If it fails again, contact technical support.
- Modify instance specifications during off-peak hours. If the modification failed in peak hours (for example, when memory or CPU usage is over 90% or write traffic surges), try again during off-peak hours.

## 5.6 Monitoring and Alarm

## 5.6.1 Does Redis Support Command Audits?

No. To ensure high-performance reads and writes, Redis does not audit commands. Commands are not printed.

## 5.6.2 What Should I Do If the Monitoring Data of a DCS Redis Instance Is Abnormal?

If you have any doubt on the monitoring data of a DCS Redis instance, you can access the instance through redis-cli and run the **INFO ALL** command to view the metrics. For details about the output of the **INFO ALL** command, see <https://redis.io/docs/latest/commands/info/>.

## 5.6.3 Why Is Available Memory of Unused DCS Instances Less Than Total Memory and Why Is Memory Usage of Unused DCS Instances Greater Than Zero?

The available memory is less than the total memory because some memory is reserved for system overhead and data persistence (supported by master/standby instances). DCS instances use a certain amount of memory for Redis-server buffers and internal data structures. This is why memory usage of unused DCS instances is greater than zero.

## 5.6.4 Why Is Used Memory Greater Than Available Memory?

For single-node and master/standby DCS instances, the used instance memory is measured by the Redis-server process. For cluster DCS instances, the used cluster memory is the sum of used memory of all shards in the cluster.

Due to the internal implementation of the open-source redis-server, the used instance memory is normally slightly higher than the available instance memory.

### Why is used\_memory higher than max\_memory?

Redis allocates memory using zmalloc. It does not check whether used\_memory exceeds max\_memory every time the memory is allocated. Instead, it checks whether the current used\_memory exceeds max\_memory at the beginning of a periodic task or command processing. If used\_memory exceeds max\_memory, eviction is triggered. Therefore, the restrictions of the max\_memory policy are not implemented in real time or rigidly. A case in which the used\_memory is greater than the max\_memory may occur occasionally.

## 5.6.5 Why Is Flow Control Triggered? How Do I Handle It?

Flow control is triggered when the traffic used by a Redis instance in a period exceeds the maximum bandwidth.

### NOTE

For details about the maximum bandwidth of an instance flavor, see [Instance Specifications](#), or refer to the page of creating an instance on the console.

Even if the bandwidth usage is low, flow control may still be triggered. The real-time bandwidth usage is reported once in each reporting period. Flow controls are

checked every second. The traffic may surge within seconds and then fall back between reporting periods. By the time the bandwidth usage is reported, it may have already restored to the normal level.

For master/standby instances:

- If flow control is always triggered when the bandwidth usage is low, there may be service microbursts or big or hot keys. In this case, check for big or hot keys.
- If the bandwidth usage remains high, the bandwidth limit may be exceeded. In this case, expand the capacity. Larger capacity supports higher bandwidth.

For cluster instances:

- If flow control is triggered only on one or a few shards, the shards may have big or hot keys.
- If flow control or high bandwidth usage occurs on all or most shards at the same time, bandwidth usage of the instance has reached the limit. In this case, expand the instance capacity.

 **NOTE**

- You can analyze big keys and hot keys on the DCS console. For details, see [Analyzing Big Keys and Hot Keys](#).
- Running commands (such as **KEYS**) that consume lots of resources may cause high CPU and bandwidth usage. As a result, flow control is triggered.

## 5.7 Data Backup, Export, and Migration

### 5.7.1 How Do I Export DCS Redis Instance Data?

- For master/standby or cluster instances:  
Perform the following operations to export the data:
  - a. On the **Backups & Restorations** page, view the backup records.
  - b. If there are no backup records, create a backup manually and download the backup file as prompted.

 **NOTE**

If your DCS instances were created a long time ago, the versions of these instances may not be advanced enough to support some new functions (such as backup and restoration). You can contact technical support to upgrade your DCS instances. After the upgrade, you can back up and restore your instances.

- For single-node instances:  
Single-node instances do not support the backup function. You can use `redis-cli` to export RDB files. This operation depends on **SYNC** command.
  - If the instance allows the **SYNC** command (such as a Redis 3.0 single-node instance), run the following command to export the instance data:  
**`redis-cli -h {source_redis_address} -p 6379 [-a password] --rdb {output.rdb}`**

- If the instance does not allow the **SYNC** command (such as a Redis 4.0 or 5.0 single-node instance), migrate the instance data to a master/standby instance and export the data by using the backup function.

## 5.7.2 Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?

- Redis 3.0  
No. On the console, backup data of a DCS Redis 3.0 instance can be exported only to AOF files. To export data to RDB files, run the following command in redis-cli:  

```
redis-cli -h {redis_address} -p 6379 [-a password] --rdb {output.rdb}
```
- Redis 4.0 and later  
Yes. DCS Redis 4.0 and later instances support AOF and RDB persistence. You can back up data to RDB and AOF files on the console and download the files.

## 5.7.3 Why Are Processes Frequently Killed During Data Migration?

Possible cause: The memory is insufficient.

Solution: Expand the memory of the server on which the migration command is executed.

## 5.7.4 Is All Data in a DCS Redis Instance Migrated During Online Migration?

Migration between single-node and master/standby instances involves the full set of data. After the migration, a given key will remain in the same DB as it was before the migration.

By contrast, a cluster instance only has one DB, which is DB0. During the migration, data in all slots of DB0 is migrated.

## 5.7.5 Do DCS Redis Instances Support Data Persistence? What Is the Impact of Persistence?

### Is Persistence Supported?

Single-node: No

Master/Standby and cluster (except single-replica clusters): Yes

### How Is Data Persisted?

- DCS Redis instances supports only AOF persistence by default. You can enable or disable persistence as required. All instances except single-node and single-replica cluster ones are created with AOF persistence enabled.
- DCS Redis instances do not support RDB persistence by default and the **save** parameter cannot be manually configured. If RDB persistence is required for a

master/standby or cluster instance of Redis 4.0 or later, you can use the backup and restoration function to back up the instance data in an RDB file and store the data in OBS.

## Disk Used for Persistence

For DCS Redis 4.0 and later instances, data is persisted to SSD disks.

## Impact of AOF Persistence

After AOF persistence is enabled, the Redis-Server process records operations in the AOF file for data persistence. This may have the following impacts:

- If the disk or I/O of the underlying compute node is faulty, the latency may increase or a master/standby switchover may occur.
- Redis-Server periodically rewrites the AOF. During a rewrite, the latency may be high for a short time. For details about the AOF rewriting rules, see [When Will AOF Rewrites Be Triggered?](#)

If DCS instances are used for application acceleration, you are advised to disable AOF persistence for higher performance and stability.

**Exercise caution when disabling AOF persistence. After it is disabled, cached data may be lost in extreme scenarios, for example, when both the master and standby nodes are faulty.**

## Configuring Redis Persistence

Set the instance configuration parameter **appendonly** to **no** to disable, or **yes** to enable AOF persistence. (Not available for single-node instances.)

For details, see [Modifying Configuration Parameters](#).

### 5.7.6 When Will AOF Rewrites Be Triggered?

AOF rewrites involve the following concepts:

- Rewrite window, which is currently 01:00 to 04:59
- Disk usage threshold, which is 50%
- Dataset memory, which is the percentage of memory that Redis dataset has used.

AOF rewrites are triggered in the following scenarios:

- If the disk usage reaches the threshold (regardless of whether the current time is within the rewrite window), rewrites will be triggered on instances whose AOF file size is larger than the dataset memory.
- If the disk usage is below the threshold and the current time is within the rewrite window, rewrites will be triggered on instances whose AOF file size is larger than the dataset memory multiplied by 1.5.
- If the disk usage is below the threshold but the current time is out of the rewrite window, rewrites will be triggered on instances whose AOF file size is larger than the instance's maximum memory multiplied by 4.5.



## 5.7.7 Online Migration with Rump

### Background

**Rump** is an open-source tool designed for migrating Redis data online. It supports migration between DBs of the same instance and between DBs of different instances.

### Migration Principles

Rump uses the **SCAN** command to acquire keys and the **DUMP/RESTORE** command to get or set values.

Featuring time complexity  $O(1)$ , **SCAN** is capable of quickly getting all keys. **DUMP/RESTORE** is used to read/write values independent from the key type.

Rump brings the following benefits:

- The **SCAN** command replaces the **KEYS** command to avoid blocking Redis.
- Any type of data can be migrated.
- **SCAN** and **DUMP/RESTORE** operations are pipelined, improving the network efficiency during data migration.
- No temporary file is involved, saving disk space.
- Buffered channels are used to optimize performance of the source server.

---

#### NOTICE

1. To cluster DCS instances, you cannot use Rump. Instead, use `redis-port` or `redis-cli`.
  2. To prevent migration command resolution errors, do not include special characters (`#@:`) in the instance password.
  3. Stop the service before migrating data. If data is kept being written in during the migration, some keys might be lost.
- 

### Step 1: Installing Rump

1. Download [Rump \(release version\)](#).

On 64-bit Linux, run the following command:

```
wget https://github.com/stickermule/rump/releases/download/0.0.3/rump-0.0.3-linux-amd64;
```

2. After decompression, run the following commands to add the execution permission:

```
mv rump-0.0.3-linux-amd64 rump;  
chmod +x rump;
```

### Step 2: Migrating Data

```
rump -from {source_redis_address} -to {target_redis_address}
```

Parameter/Option description:

- *{source\_redis\_address}*  
Source Redis instance address, in the format of redis://[user:password@]host:port/db. **[user:password@]** is optional. If the instance is accessed in password-protected mode, you must specify the password in the RFC 3986 format. **user** can be omitted, but the colon (:) cannot be omitted. For example, the address may be **redis://:mypassword@192.168.0.45:6379/1**. **db** is the sequence number of the database. If it is not specified, the default value is 0.
- *{target\_redis\_address}*  
Address of the target Redis instance, in the same format as the source.  
In the following example, data in DBO of the source Redis is migrated to the target Redis whose connection address is 192.168.0.153. **\*\*\*\*\*** stands for the password.

```
[root@ecs ~]# ./rump -from redis://127.0.0.1:6379/0 -to redis://:*****@192.168.0.153:6379/0
.Sync done.
[root@ecs ~]#
```

## 5.8 Big/Hot Key Analysis and Expired Key Scan

### 5.8.1 What Are Big Keys and Hot Keys?

Term	Definition
Big key	<p>There are two types of big keys:</p> <ul style="list-style-type: none"> <li>• Keys that have a large value. If the size of a single String key exceeds 10 KB, or if the size of all elements of a key combined exceeds 50 MB, the key is defined as a big key.</li> <li>• Keys that have a large number of elements. If the number of elements in a key exceeds 5000, the key is defined as a big key.</li> </ul>
Hot key	<p>A hot key is most frequently accessed, or consumes significant resources. For example:</p> <ul style="list-style-type: none"> <li>• In a cluster instance, a shard processes 10,000 requests per second, among which 3000 are performed on the same key.</li> <li>• In a cluster instance, a shard uses a total of 100 Mbits/s inbound and outbound bandwidth, among which 80 Mbits/s is used by the <b>HGETALL</b> operation on a Hash key.</li> </ul>

## 5.8.2 What Is the Impact of Big Keys or Hot Keys?

Category	Impact
Big key	<p><b>Instance specifications fail to be modified.</b></p> <p>Specification modification of a Redis Cluster instance involves rebalancing (data migration between nodes). Redis has a limit on key migration. If the instance has any single key bigger than 512 MB, the modification will fail when big key migration between nodes times out. The bigger the key, the more likely the migration will fail.</p>
	<p><b>Data migration fails.</b></p> <p>During data migration, if a key has many elements, other keys will be blocked and will be stored in the memory buffer of the migration ECS. If they are blocked for a long time, the migration will fail.</p>
	<p><b>Cluster shards are unbalanced.</b></p> <ul style="list-style-type: none"> <li>• The memory usage of shards is unbalanced. For example, if a shard uses a large memory or even uses up the memory, keys on this shard are evicted, and resources of other shards are wasted.</li> <li>• The bandwidth usage of shards is unbalanced. For example, flow control is repeatedly triggered on a shard.</li> </ul>
	<p><b>Latency of client command execution increases.</b></p> <p>Slow operations on a big key block other commands, resulting in a large number of slow queries.</p>
	<p><b>Flow control is triggered on the instance.</b></p> <p>Frequently reading data from big keys exhausts the outbound bandwidth of the instance, triggering flow control. As a result, a large number of commands time out or slow queries occur, affecting services.</p>
	<p><b>Master/standby switchover is triggered.</b></p> <p>If the high-risk <b>DEL</b> operation is performed on a big key, the master node may be blocked for a long time, causing a master/standby switchover.</p>
Hot key	<p><b>Cluster shards are unbalanced.</b></p> <p>If only the shard where the hot key is located is busy processing service queries, there may be performance bottlenecks on a single shard, and the compute resources of other shards may be wasted.</p>

Category	Impact
	<p><b>CPU usage surges.</b></p> <p>A large number of operations on hot keys may cause high CPU usage. If the operations are on a single cluster shard, the CPU usage of the shard where the hot key is located will surge. This will slow down other requests and the overall performance. If the service volume increases sharply, a master/standby switchover may be triggered.</p>
	<p><b>Cache breakdown may occur.</b></p> <p>If Redis cannot handle the pressure on hot keys, requests will hit the database. The database may break down as its load increases dramatically, affecting other services.</p>

### 5.8.3 How Do I Avoid Big Keys and Hot Keys?

- **Keep the size of Strings within 10 KB and the quantity of Hashes, Lists, Sets, or Zsets within 5000.**
- When naming keys, use the service name abbreviation as the prefix and do not use special characters such as spaces, line brakes, single or double quotation marks, and other escape characters.
- Do not rely too much on Redis transactions.
- The performance of short connections ("connect" in Redis terminology) is poor. Use clients with connection pools.
- Do not enable data persistence if you use Redis just for caching and can tolerate data loss.
- For details about how to optimize big keys and hot keys, see the following table.

Category	Method
Big key	<p><b>Split big keys.</b></p> <p>Scenarios:</p> <ul style="list-style-type: none"> <li>• <b>If the big key is a String</b>, you can split it into several key-value pairs and use <b>MGET</b> or a pipeline consisting of multiple <b>GET</b> operations to obtain the values. In this way, the pressure of a single operation can be split. For a cluster instance, key-value pairs can be automatically distributed to multiple shards, reducing the impact on a single shard.</li> <li>• <b>If the big key contains multiple elements, and the elements must be operated together</b>, the big key cannot be split. You can remove the big key from Redis and store it on other storage media instead. This scenario should be avoided by design.</li> <li>• <b>If the big key contains multiple elements, and only some elements need to be operated each time</b>, separate the elements. Take a Hash key as an example. Each time you run the <b>HGET</b> or <b>HSET</b> command, the result of the hash value modulo <math>N</math> (customized on the client) determines which key the field falls on. This algorithm is similar to that used for calculating slots in Redis Cluster.</li> </ul> <p><b>Store big keys on other storage media.</b></p> <p>If a big key cannot be split, it is not suitable to be stored in Redis. You can store it on other storage media, and delete the big key from Redis.</p> <p><b>CAUTION</b></p> <p>Do not use the <b>DEL</b> command to delete big keys. Otherwise, Redis may be blocked or even a master/standby switchover may occur.</p>
Hot key	<p><b>Use the client cache or local cache.</b></p> <p>If you know what keys are frequently used, you can design a two-level cache architecture (client/local cache and remote Redis). Frequently used data is obtained from the local cache first. The local cache and remote cache are updated with data writes at the same time. In this way, the read pressure on frequently accessed data can be separated. This method is costly because it requires changes to the client architecture and code.</p> <p><b>Design a circuit breaker or degradation mechanism.</b></p> <p>Hot keys can easily result in cache breakdown. During peak hours, requests are passed through to the backend database, causing service avalanche. To ensure availability, the system must have a circuit breaker or degradation mechanism to limit the traffic and degrade services if breakdown occurs.</p>

## 5.8.4 How Do I Analyze the Hot Keys of a DCS Redis 3.0 Instance?

DCS for Redis 3.0 does not support hot key analysis on the console. Alternatively, you can use the following methods to analyze hot keys:

- Method 1: Analyze the service structure and service implementation to discover possible hot keys.  
For example, hot keys can be easily found in the service code during flash sales or user logins.  
Advantage: Simple and easy to implement.  
Disadvantage: Requires familiarity with the service code. In addition, the analysis become more difficult as the service scenarios become more complex.
- Method 2: Collect key access statistics in the client code to discover hot keys.  
Disadvantage: Requires intrusive code modification.
- Method 3: Capture and analyze packets.  
Advantage: Simple and easy to implement.

## 5.8.5 How Do I Detect Big Keys and Hot Keys in Advance?

Method	Description
Through <b>Big Key Analysis</b> and <b>Hot Key Analysis</b> on the DCS console	See <a href="#">Analyzing Big Keys and Hot Keys</a> .
By using the <b>bigkeys</b> and <b>hotkeys</b> options on redis-cli	<ul style="list-style-type: none"> <li>• redis-cli uses the <b>bigkeys</b> option to traverse all keys in a Redis instance and returns the overall key statistics and the biggest key of six data types: Strings, Lists, Hashes, Sets, Zsets, and Streams. The command is <b>redis-cli -h &lt;Instance connection address&gt; -p &lt;Port number&gt; -a &lt;Password&gt; --bigkeys</b>.</li> <li>• In Redis 4.0 and later, you can use the <b>hotkeys</b> option to quickly find hot keys in redis-cli. Run this command during service running to find hot keys: <b>redis-cli -h &lt;Instance connection address&gt; -p &lt;Port number&gt; -a &lt;Password&gt; --hotkeys</b>. The hot key details can be obtained from the summary part in the returned result.</li> </ul>

## 5.8.6 How Does DCS Delete Expired Keys?

### Question

What are the rules for scheduled deletion of expired keys on a daily basis? Can I customize the rules?

## Mechanisms for Deleting Expired Keys

- Lazy free deletion: The deletion strategy is controlled in the main I/O event loop. Before a read/write command is executed, a function is called to check whether the key to be accessed has expired. If it has expired, it will be deleted and a response will be returned indicating that the key does not exist. If the key has not expired, the command execution resumes.
- Scheduled deletion: A time event function is executed at certain intervals. Each time the function is executed, a random collection of keys are checked, and expired keys are deleted.

### NOTE

To avoid prolonged blocks on the Redis main thread, not all keys are checked in each time event. Instead, a random collection of keys are checked each time. As a result, the memory used by expired keys cannot be released quickly.

## Solutions

- Configure scheduled hot key analysis tasks by referring to [Hot Key Analysis](#), or use the **SCAN** command to traverse all keys on a scheduled basis and remove expired keys from the memory.
- Configure a scheduled task to scan all master nodes of the instance. All keys will be scanned, and Redis will determine whether the keys have expired. Expired keys will be released. For details, see [Scanning Expired Keys](#).

## How Do I Know Which Expired Keys Have Been Deleted?

Deleted expired keys cannot be queried.

## 5.8.7 How Long Are Keys Stored? How Do I Set Key Expiration?

- Key storage duration
  - Keys that do not have an expiration are stored permanently.
  - Keys that have an expiration are deleted after they expire. For details, see [Scanning Expired Keys](#).
  - To remove the expiration set for a key, run the **PERSIST** command.
- Setting key expiration

You can run the **EXPIRE** or **PEXPIRE** command to set the key expiration time. For example, if you run **expire key1 100**, key1 will expire in 100 seconds. If you run **pexpire key2 1800**, key2 will expire in 1800 milliseconds.

**EXPIRE** sets key expiration in seconds, and **PEXPIRE** sets key expiration in milliseconds.

## 5.9 Master/Standby Switchover

### 5.9.1 When Does a Master/Standby Switchover Occur?

A master/standby switchover may occur in the following scenarios:

- A master/standby switchover operation is initiated on the DCS Console.
- If the master node of a master/standby instance fails, a master/standby switchover will be triggered.

For example, running commands that consume a lot of resources, such as **KEYS** commands, will cause CPU usage to spike and as result triggers a master/standby switchover.

- If you restart a master/standby instance on the DCS console, a master/standby switchover will be triggered.
- If you scale up a single-node or master/standby instance, a master/standby switchover will be triggered.

During scale-up, a new standby node with the new specifications is created. After full and incremental data on the master node is synchronized to the standby node, a master/standby switchover is performed and the original node is deleted.

After a master/standby switchover occurs, you will receive a notification. Check whether the client services are running properly. If not, check whether the TCP connection is normal and whether it can be re-established after the master/standby switchover to restore the services.

## 5.9.2 How Does Master/Standby Switchover Affect Services?

If a fault occurs in a master/standby or cluster DCS instance, a failover is triggered automatically. Services may be interrupted for less than half a minute during exception detection and failover.

## 5.9.3 Does the Client Need to Switch the Connection Address After a Master/Standby Switchover?

No. If the master fails, the standby node will be promoted to master and takes the original IP address.

## 5.9.4 How Does Redis Master/Standby Replication Work?

Redis master/standby instances are also called master/slave instances. Generally, updates to the master cache node are automatically and asynchronously replicated to the standby cache node. This means that data in the standby cache node may not always be consistent with data in the master cache node. The inconsistency is typically seen when the I/O write speed of the master node is faster than the synchronization speed of the standby node or a network latency occurs between the master and standby nodes. If a failover happens when some data is not yet replicated to the standby node, such data may be lost after the failover.