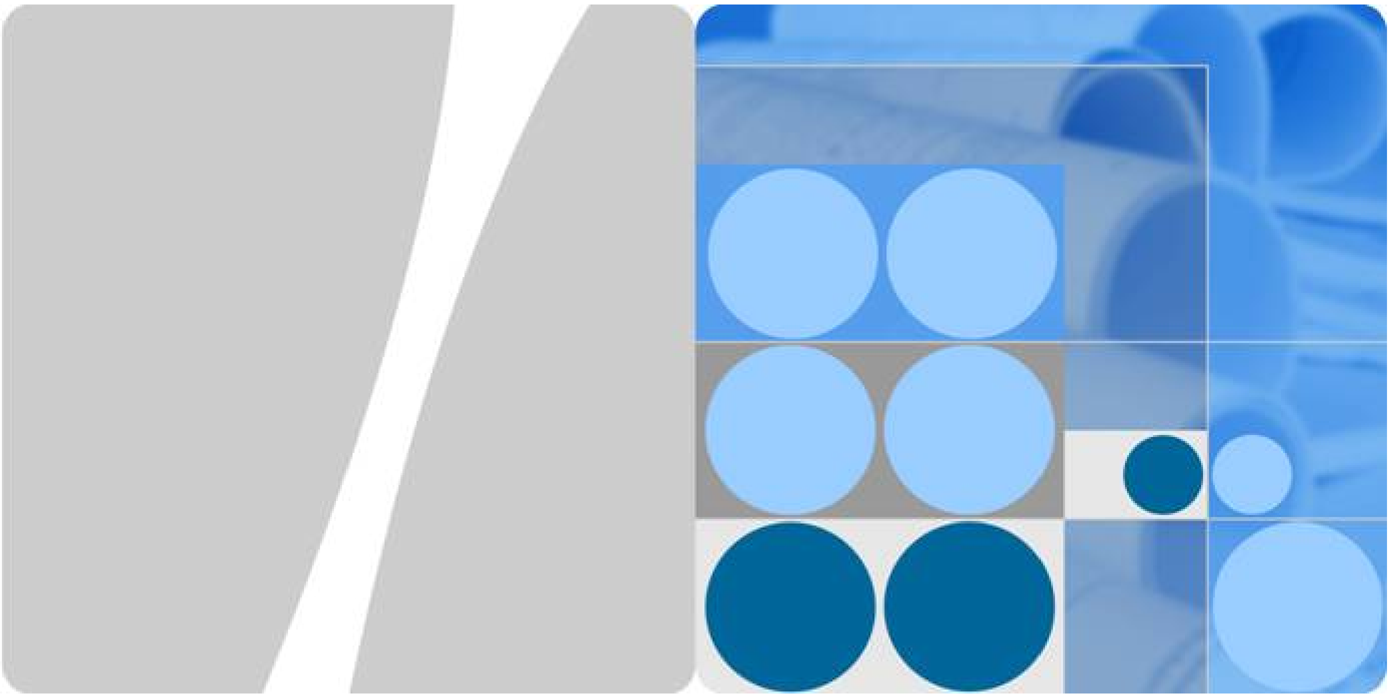


Distributed Cache Service

User Guide

Date **2020-09-30**

Contents

1 Service Overview

1.1 What Is DCS?

Distributed Cache Service (DCS) is an online, distributed, 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 1 TB. DCS instances can be created with just a few clicks on the console, without requiring you to prepare servers.

  DCS Redis 4.0 and 5.0 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), Cloud Eye, and Cloud Trace Service (CTS).

  Master/Standby and cluster instances can be deployed within an availability zone (AZ) or across AZs.

- 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

Instance type	DCS for Redis provides the following three types of instances to suit different service scenarios: 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. 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. Cluster: Each cluster DCS instance consists of multiple **shards** and each shard includes a master node and zero or multiple replicas. Shards are not visible to users. If the master node fails, a standby node in the same shard takes over.
Instance specification	DCS for Redis provides instances of different specifications, ranging from 128 MB to 1 TB.
Redis version	DCS instances are compatible with open-source Redis 3.0, 4.0, and 5.0.
Underlying architecture	**Standard** Redis based on VMs: supports up to 100,000 queries per second (QPS) at a single node.
High availability (HA) and DR	Master/Standby and cluster DCS Redis instances can be deployed across AZs in the same region with physically isolated power supplies and networks.

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 100,000 QPS per node and allows you to implement locking using simple commands such as **SET**, **GET**, **DEL**, and **RPUSH** 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

Three Redis versions are available for single-node DCS Redis instances: Redis 3.0, Redis 4.0, and Redis 5.0.

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 100,000.

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.

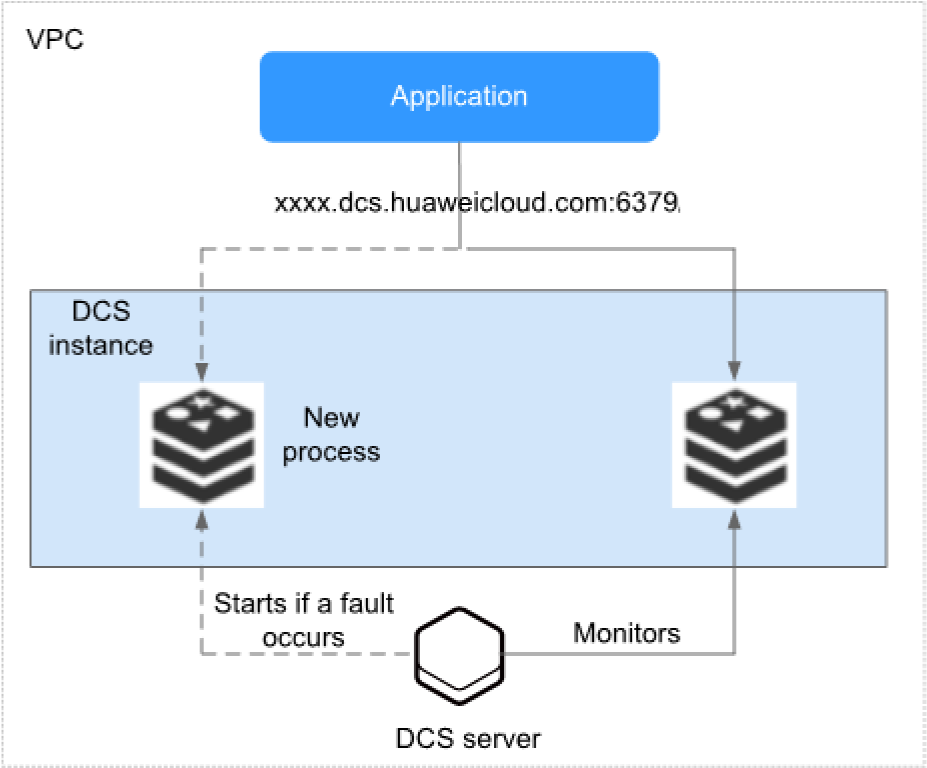
Architecture

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 5.0 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**

  The VPC where all nodes of the instance are run.

  NOTE

  For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

  For details, see **Security Group Configurations**.

- **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 examples of accessing DCS instances, see **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. Three Redis versions are available for master/standby DCS Redis instances: Redis 3.0, Redis 4.0, and Redis 5.0.

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 DCS instance.

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

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 standby node takes over within 30 seconds, without requiring any service interruptions or manual operations.

4. **DR policies**

   Each master/standby instance can be deployed across AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve HA for both data and applications.

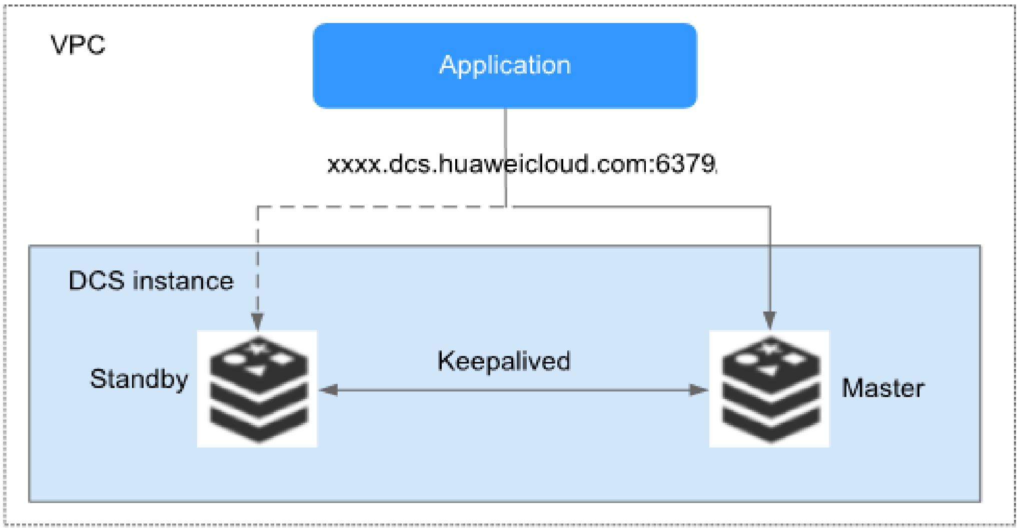
Architecture

Figure 1-2 shows the architecture of master/standby 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 5.0 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 1-2 Master/Standby DCS instance architecture



Architecture description:

- **VPC**

  The VPC where all nodes of the instance are run.

  NOTE

  For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

  For details, see **Security Group Configurations**.

- **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 examples of accessing DCS instances, see **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 instances are accessed through port 6379 by default.

1.3.3 Proxy Cluster Redis

DCS provides two types of cluster Redis instances: Proxy Cluster and Redis Cluster. Proxy Cluster uses Linux Virtual Server (LVS) and proxies. Redis Cluster is the native distributed implementation of Redis. Proxy Cluster instances are compatible with Redis 3.0, while Redis Cluster instances are compatible with Redis 4.0 and 5.0.

This section describes Proxy Cluster DCS Redis 3.0 instances.

Proxy Cluster DCS Redis 3.0 Instances

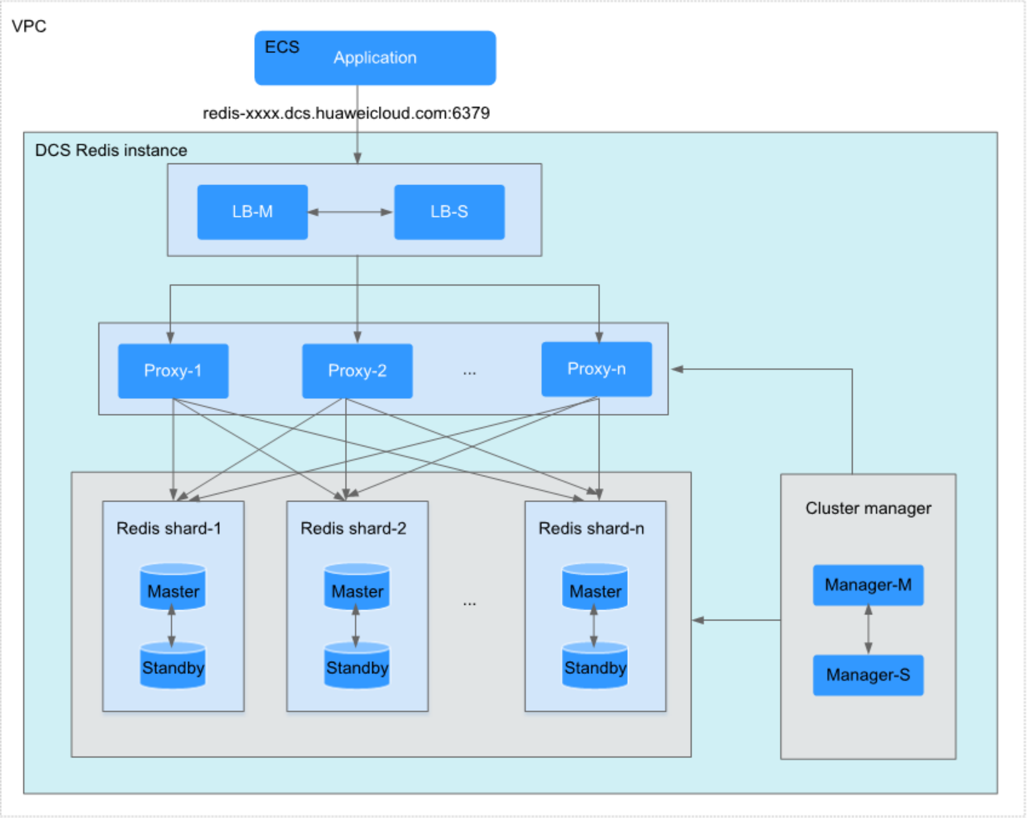
The specifications of Proxy Cluster DCS Redis 3.0 instances range from 64 GB to 1024 GB, meeting requirements for **millions of concurrent connections** and **massive data cache**. Distributed data storage and access is implemented by DCS, without requiring development or maintenance.

Each Proxy Cluster instance consists of load balancers, proxies, cluster managers, and **shards**.

Table 1-2 Specifications of Proxy Cluster DCS Redis 3.0 instances

Total Memory	Proxies	Shards
64 GB	3	8
128 GB	6	16
256 GB	8	32
512 GB	16	64
1024 GB	32	128

Figure 1-3 Proxy Cluster DCS Redis instance architecture



Architecture description:

- **VPC**

  The VPC where all nodes of the instance are run.

  NOTE

  For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

  For details, see **Security Group Configurations**.

- **Application**

  The client used to access the instance.

  DCS Redis instances can be accessed through open-source clients. For examples of accessing DCS instances, see **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.

1.3.4 Redis Cluster

DCS provides two types of cluster Redis instances: Proxy Cluster and Redis Cluster. Proxy Cluster uses Linux Virtual Server (LVS) and proxies. Redis Cluster is the native distributed implementation of Redis. Proxy Cluster instances are compatible with Redis 3.0, while Redis Cluster instances are compatible with Redis 4.0 and 5.0.

This section describes Redis Cluster DCS Redis 4.0 and 5.0 instances.

Redis Cluster DCS Redis 4.0 and 5.0 Instances

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 1-3 lists the shard specification for different instance specifications.

Specification per shard=Instance specification/Number of shards. For example, if a 48 GB instance has 6 shards, the specification of each shard is 48 GB/6 = 8 GB.

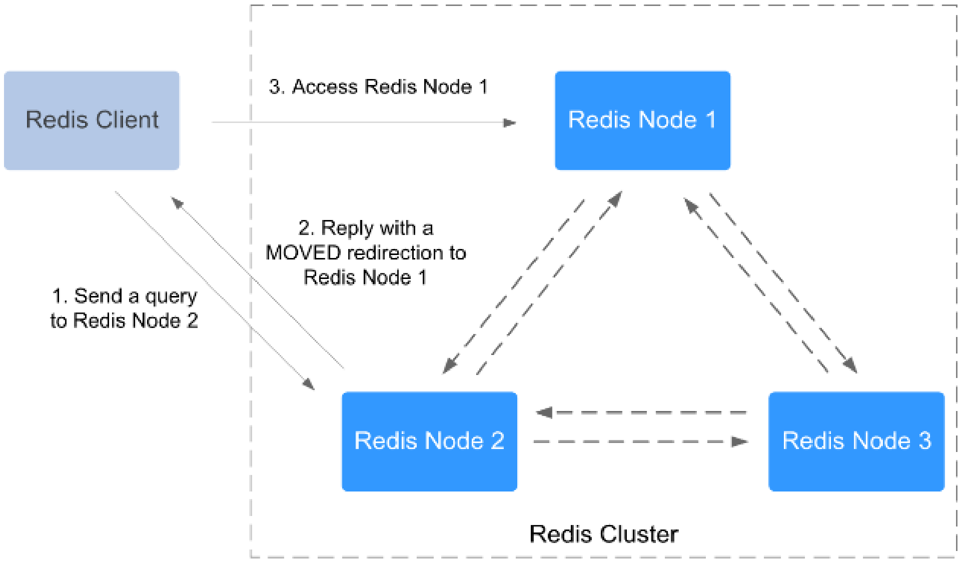
Table 1-3 Specifications of Redis Cluster DCS instances

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
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.

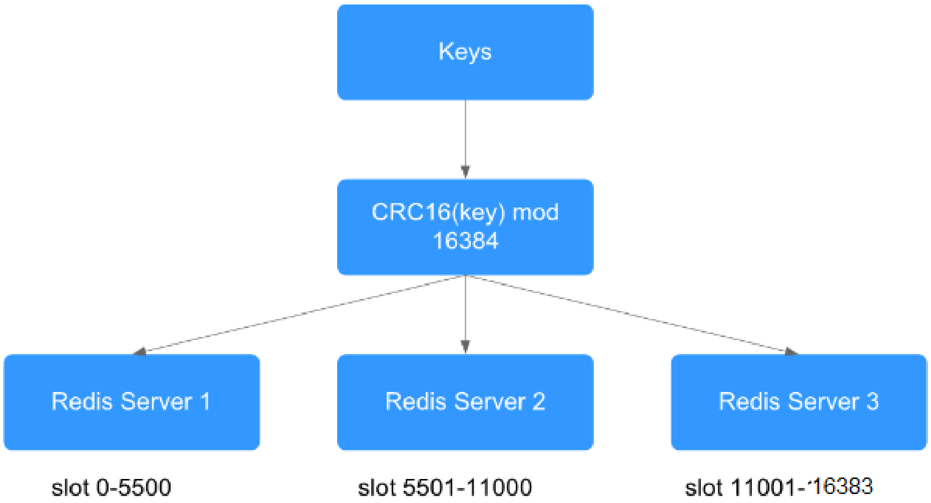
  Figure 1-4 Distributed architecture of Redis Cluster



- 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 1-5 Redis Cluster presharding



1.4 DCS Instance Specifications

1.4.1 Redis 3.0 Instance Specifications

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.

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 1-4 Specifications of single-node DCS Redis 3.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/512	50,000
4	3.2	5000/50,000	64/1536	100,000
8	6.8	5000/50,000	64/1536	100,000
16	13.6	5000/50,000	85/3072	100,000
32	27.2	5000/50,000	85/3072	100,000
64	58.2	5000/60,000	128/5120	100,000

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 1-5 Specifications of master/standby DCS Redis 3.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/512	50,000
4	3.2	5000/50,000	64/1536	100,000
8	6.4	5000/50,000	64/1536	100,000
16	12.8	5000/50,000	85/3072	100,000
32	25.6	5000/50,000	85/3072	100,000
64	51.2	5000/60,000	128/5120	100,000

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 1-6 Specifications of Proxy Cluster DCS Redis 3.0 instances

Specification (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
64	64	90,000/90,000	600/5120	500,000
128	128	180,000/180,000	600/5120	500,000
256	256	240,000/240,000	600/5120	500,000
512	512	480,000/480,000	600/5120	500,000
1024	1024	960,000/960,000	600/5120	500,000

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.
- 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, and Redis Cluster types are available.

Single-Node Instances

Table 1-7 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)
0.125	0.125	10,000/10,000	40/40	80,000
0.25	0.25	10,000/10,000	80/80	80,000
0.5	0.5	10,000/10,000	80/80	80,000
1	1	10,000/50,000	80/80	80,000
2	2	10,000/50,000	128/128	80,000
4	4	10,000/50,000	192/192	80,000
8	8	10,000/50,000	192/192	100,000
16	16	10,000/50,000	256/256	100,000
24	24	10,000/50,000	256/256	100,000
32	32	10,000/50,000	256/256	100,000
48	48	10,000/50,000	256/256	100,000
64	64	10,000/50,000	384/384	100,000

Master/Standby Instances

Table 1-8 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)
0.125	0.125	10,000/10,000	40/40	80,000
0.25	0.25	10,000/10,000	80/80	80,000
0.5	0.5	10,000/10,000	80/80	80,000
1	1	10,000/50,000	80/80	80,000

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	2	10,000/50,000	128/128	80,000
4	4	10,000/50,000	192/192	80,000
8	8	10,000/50,000	192/192	100,000
16	16	10,000/50,000	256/256	100,000
24	24	10,000/50,000	256/256	100,000
32	32	10,000/50,000	256/256	100,000
48	48	10,000/50,000	256/256	100,000
64	64	10,000/50,000	384/384	100,000

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.

Table 1-9 Specifications of Redis Cluster DCS Redis 4.0 or 5.0 instances

Specification (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
4	4	3	30,000 /150,000	2304/2304	240,000
8	8	3	30,000 /150,000	2304/2304	240,000

Specification (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
16	16	3	30,000 /150,000	2304/2304	240,000
24	24	3	30,000 /150,000	2304/2304	300,000
32	32	3	30,000 /150,000	2304/2304	300,000
48	48	6	60,000 /300,000	4608/4608	> 300,000
64	64	8	80,000 /400,000	6144/6144	500,000
96	96	12	120,000 /600,000	9216/9216	> 500,000
128	128	16	160,000 /800,000	12,288/12,288	1,000,000
192	192	24	240,000 /1,200,000	18,432/18,432	> 1,000,000
256	256	32	320,000 /1,600,000	24,576/24,576	> 2,000,000
384	384	48	480,000 /2,400,000	36,864/36,864	> 2,000,000
512	512	64	640,000 /3,200,000	49,152/49,152	> 2,000,000
768	768	96	960,000 /4,800,000	73,728/73,728	> 2,000,000
1024	1024	128	1,280,000 /6,400,000	98,304/98,304	> 2,000,000

1.5 Command Compatibility

1.5.1 Redis 3.0 Commands

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 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 3.0**.
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see **Command Restrictions for Cluster Instances**.
- 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 1-10 Commands supported by DCS Redis 3.0 instances (1/2)

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

Key	String	Hash	List	Set	Sorted set	Server
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSISI	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT GETNAME
RENAMENX	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	-	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJIECT	PSETEX	-	RPUSH	-	ZSCAN	-
-	SET	-	RPUSHX	-	ZRANGEBYLEX	-
-	SETBIT	-	-	-	-	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-

Table 1-11 Commands supported by DCS Redis 3.0 instances (2/2)

HyperLogLog	Pub/Sub	Transaction	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 1-12 Redis commands disabled in single-node and master/standby Redis 3.0 instances

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

Table 1-13 Redis commands disabled in Proxy Cluster Redis 3.0 instances

Key	Server	List	Transaction	Connection	Cluster	codis
MIGRATE	SLAVEOF	BLPOP	DISCARD	SELECT	CLUSTER	TIME
SELECT	SHUTDOWN	BRPOP	EXEC	TIME	-	SLOTSINFO
-	LASTSAVE	BRPOPLPUSH	MULTI	-	-	SLOTSDEL
-	DEBUG commands	-	UNWATCH	-	-	SLOTSMGRTSLOT
-	COMMAND	-	WATCH	-	-	SLOTSMGRTONE
-	SAVE	-	-	-	-	SLOTSCHECK
-	BGSAVE	-	-	-	-	SLOTSMGRTTAGSLOT
-	BGREWRITEAOF	-	-	-	-	SLOTSMGRTTAGONE
-	SYNC	-	-	-	-	-
-	PSYNC	-	-	-	-	-
-	MONITOR	-	-	-	-	-
-	CLIENT commands	-	-	-	-	-
-	OBJECT	-	-	-	-	-
-	ROLE	-	-	-	-	-

1.5.2 Redis 4.0 Commands

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. 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 for Cluster Instances**.
- Some Redis commands have usage restrictions, which are described in **Other Command Usage Restrictions**.

Commands Supported by DCS for Redis 4.0

Table 1-14 and **Table 1-15** list the Redis commands supported by 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.

Table 1-14 Commands supported by DCS Redis 4.0 instances (1/2)

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
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSISI	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT GETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET

Key	String	Hash	List	Set	Sorted set	Server
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJIECT	PSETEX	-	RPUSH	SPOP	ZSCAN	MEMORY
-	SET	-	RPUSHX	-	ZRANGEBYLEX	-
-	SETBIT	-	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

Table 1-15 Commands supported by DCS Redis 4.0 instances (2/2)

HyperLogLog	Pub/Sub	Transaction	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 4.0

The following lists commands disabled by DCS for Redis 4.0.

Table 1-16 Redis commands disabled in single-node and master/standby Redis 4.0 instances

Key	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

Table 1-17 Redis commands disabled in Redis Cluster Redis 4.0 instances

Key	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	COMMAND	CLUSTER SETSLOT
-	SAVE	CLUSTER BUMPEPOCH
-	BGSAVE	CLUSTER SAVECONFIG
-	BGREWRITEAOF	CLUSTER FORGET
-	SYNC	CLUSTER REPLICATE
-	PSYNC	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH

Key	Server	Cluster
-	-	CLUSTER RESET

1.5.3 Redis 5.0 Commands

DCS for Redis 5.0 is developed based on Redis 5.0.9 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 for Cluster Instances**.
- Some Redis commands have usage restrictions, which are described in **Other Command Usage Restrictions**.

Commands Supported by DCS for Redis 5.0

The following lists commands supported by DCS for Redis 5.0.

Table 1-18 Commands supported by DCS Redis 5.0 instances (1/2)

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
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSISI	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL

Key	String	Hash	List	Set	Sorted set	Server
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT GETNAME
RENAMENX	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	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJIECT	PSETEX	-	RPUSH	SPOP	ZSCAN	MEMORY
-	SET	-	RPUSHX	-	ZRANGEBYLEX	-
-	SETBIT	-	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	ZPOPMIN	-
-	SETNX	-	-	-	ZPOPMAX	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

Table 1-19 Commands supported by DCS Redis 5.0 instances (2/2)

HyperLogLog	Pub/Sub	Transaction	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD

HyperLogLog	Pub/Sub	Transaction	Connection	Scripting	Geo	Stream
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	-	-	-	XLEN
-	-	-	-	-	-	XPENDING
-	-	-	-	-	-	XRANGE
-	-	-	-	-	-	XREAD
-	-	-	-	-	-	XREADGROUP
-	-	-	-	-	-	XREVRANGE
-	-	-	-	-	-	XTRIM

Commands Disabled by DCS for Redis 5.0

The following lists commands disabled by DCS for Redis 5.0.

Table 1-20 Redis commands disabled in single-node and master/standby Redis 5.0 instances

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

Key	Server
-	SYNC
-	PSYNC

Table 1-21 Redis commands disabled in Redis Cluster Redis 5.0 instances

Key	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	COMMAND	CLUSTER SETSLOT
-	SAVE	CLUSTER BUMPEPOCH
-	BGSAVE	CLUSTER SAVECONFIG
-	BGREWRITEAOF	CLUSTER FORGET
-	SYNC	CLUSTER REPLICATE
-	PSYNC	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.4 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 Redis 4.0 and 5.0 support Web CLI.

Commands Supported in Web CLI

The following lists the commands supported when the using Web CLI.

Table 1-22 Commands supported by Web CLI (1/2)

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
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSISI	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	CLIENT KILL
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT LIST
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT GETNAME
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT SETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CONFIG GET
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	MONITOR
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	SLOWLOG
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	ROLE
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	SWAPDB
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	MEMORY
OBJIECT	PSETEX	-	RPUSH	SPOP	ZSCAN	-
-	SET	-	RPUSHX	-	ZRANGEBYLEX	-
-	SETBIT	-	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-

Key	String	Hash	List	Set	Sorted set	Server
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

Table 1-23 Commands supported by Web CLI (2/2)

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

Commands Disabled in Web CLI

The following lists the commands disabled when the using Web CLI.

Table 1-24 Redis commands disabled in Web CLI for single-node and master/standby instances

Key	Server	Transaction
MIGRATE	SLAVEOF	UNWATCH
WAIT	SHUTDOWN	REPLICAOF
-	DEBUG commands	-
-	CONFIG SET	-
-	CONFIG REWRITE	-
-	CONFIG RESETSTAT	-

Key	Server	Transaction
-	SAVE	-
-	BGSAVE	-
-	BGREWRITEAOF	-
-	COMMAND	-
-	KEYS	-
-	MONITOR	-
-	SYNC	-
-	PSYNC	-

Table 1-25 Redis commands disabled in Web CLI for Redis Cluster instances

Key	Server	Transaction	Cluster
MIGRATE	SLAVEOF	UNWATCH	CLUSTER MEET
WAIT	SHUTDOWN	REPLICAOF	CLUSTER FLUSHSLOTS
-	DEBUG commands	-	CLUSTER ADDSLOTS
-	CONFIG SET	-	CLUSTER DELSLOTS
-	CONFIG REWRITE	-	CLUSTER SETSLOT
-	CONFIG RESETSTAT	-	CLUSTER BUMPEPOCH
-	SAVE	-	CLUSTER SAVECONFIG
-	BGSAVE	-	CLUSTER FORGET
-	BGREWRITEAOF	-	CLUSTER REPLICATE
-	COMMAND	-	CLUSTER COUNT-FAILURE-REPORTS
-	KEYS	-	CLUSTER FAILOVER
-	MONITOR	-	CLUSTER SET-CONFIG-EPOCH
-	SYNC	-	CLUSTER RESET
-	PSYNC	-	-

1.5.5 Command Restrictions for Cluster Instances

Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see **Table 1-26**.

Table 1-26 Redis commands restricted in cluster DCS instances.

Category	Description
Set	
SINTER	Returns the members of the set resulting from the intersection of all the given sets.
SINTERSTORE	Equal to **SINTER**, but instead of returning the result set, it is stored in *destination*.
SUNION	Returns the members of the set resulting from the union of all the given sets.
SUNIONSTORE	Equal to **SUNION**, but instead of returning the result set, it is stored in *destination*.
SDIFF	Returns the members of the set resulting from the difference between the first set and all the successive sets.
SDIFFSTORE	Equal to **SDIFF**, but instead of returning the result set, it is stored in *destination*.
SMOVE	Moves **member** from the set at **source** to the set at *destination*.
Sorted Set	
ZUNIONSTORE	Computes the union of *numkeys* sorted sets given by the specified keys.
ZINTERSTORE	Computes the intersection of *numkeys* sorted sets given by the specified keys.
HyperLogLog	
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.
Keys	
RENAME	Renames *key* to *newkey*.
RENAMENX	Renames *key* to *newkey* if *newkey* does not yet exist.

Category	Description
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 source, and pushes the element at the first element (head) of the list stored at *destination*.
String	
MSETNX	Sets the given keys to their respective values.

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.

1.5.6 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.

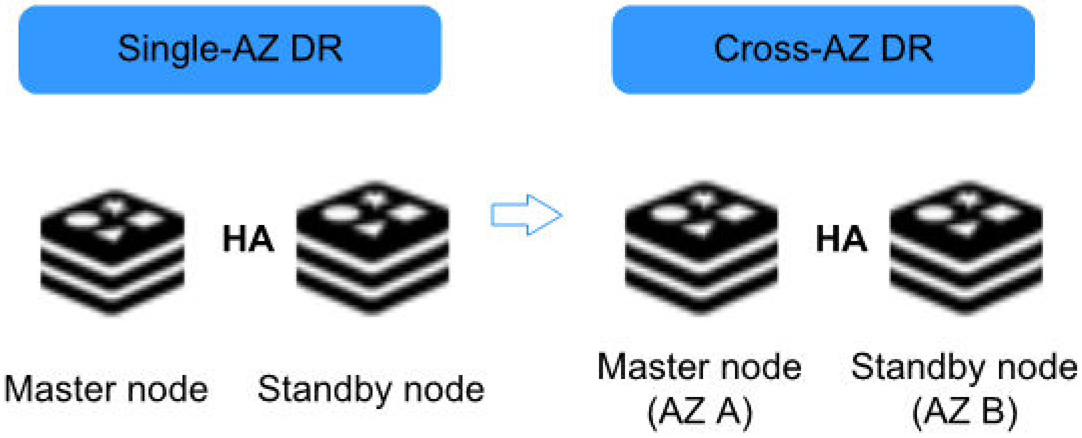
Other Restrictions

- 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 DCS Disaster Recovery

Whether you use DCS as the frontend cache or backend data store, DCS is always ready to ensure data reliability and service availability. The following figure shows the evolution of DCS DR architectures.

Figure 1-6 DCS DR architecture evolution



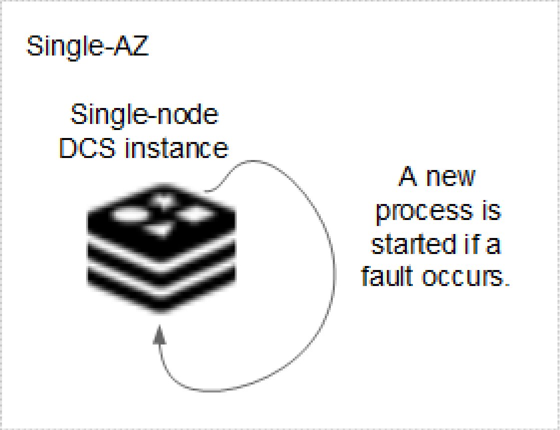
To meet the reliability requirements of your data and services, you can choose to deploy your DCS instance within a single AZ or across AZs.

Single-AZ HA

Single-AZ deployment means deploying 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.

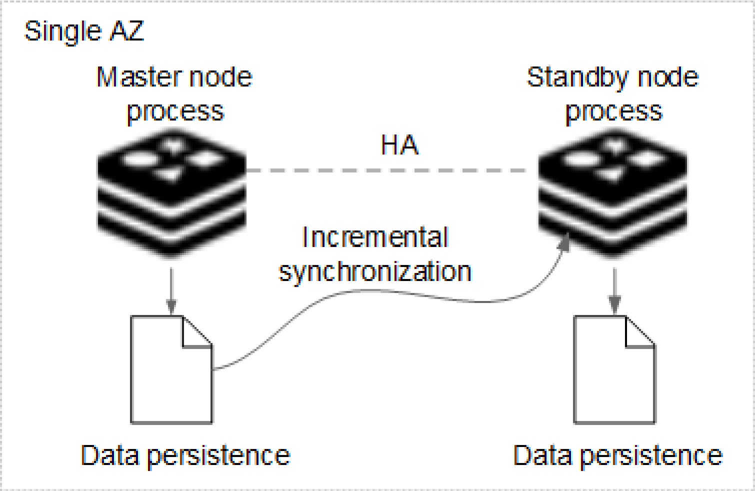
Single-node DCS instance: When DCS detects a process fault, a new process is started to ensure service HA.

Figure 1-7 HA for a single-node DCS instance deployed within an AZ



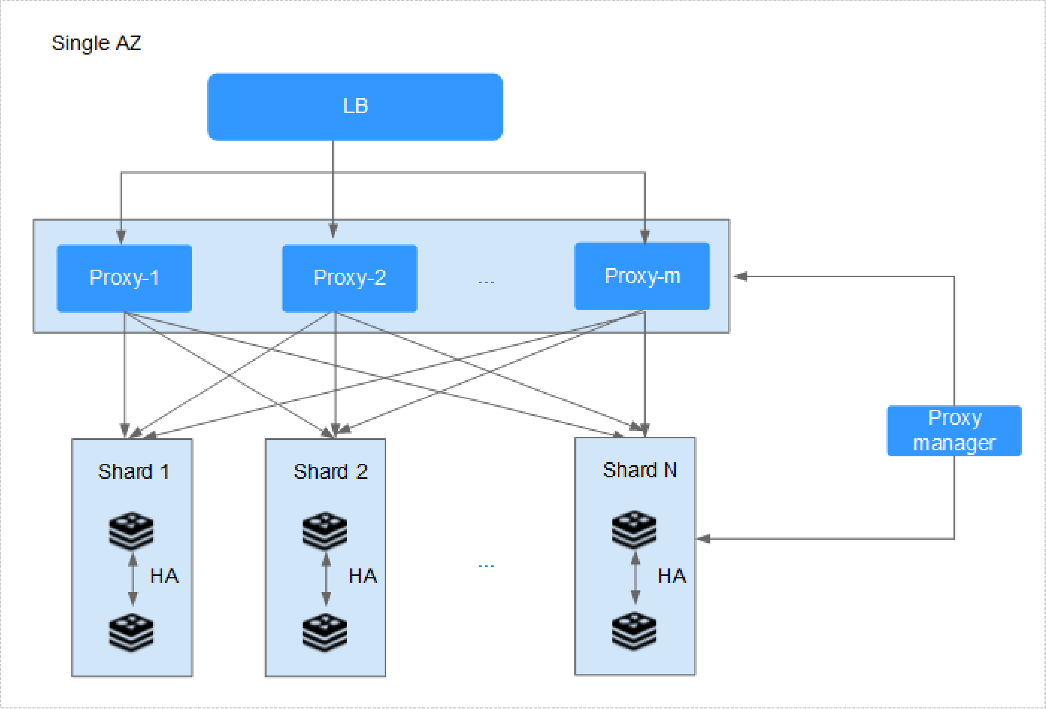
Master/Standby DCS instance: Data is persisted to disk in the master node and incrementally synchronized and persisted to the standby node, achieving hot standby and data persistence.

Figure 1-8 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 in both nodes.

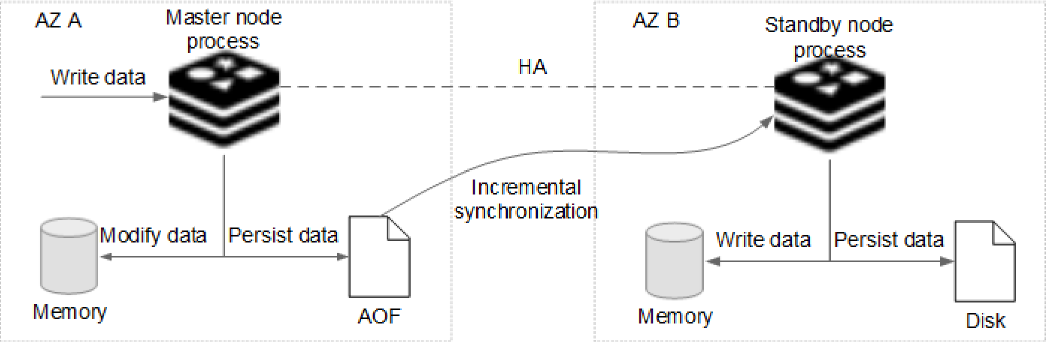
Figure 1-9 HA for a cluster DCS instance deployed within an AZ



Cross-AZ DR

The master and standby nodes of a master/standby or cluster DCS instance can be deployed across AZs (in different equipment rooms). Power supplies and networks of different AZs are physically isolated. When a fault occurs in the AZ where the master node is deployed, the standby node connects to the client and takes over data read and write operations.

Figure 1-10 Cross-AZ deployment of a master/standby DCS instance



NOTE

This mechanism applies in a similar way to a cluster DCS instance, in which each shard (process) is deployed across AZs.

When creating a master/standby DCS instance, select a standby AZ that is different from the primary AZ.

NOTE

You can also deploy your application across AZs to ensure both data reliability and service availability in the event of power supply or network disruptions.

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, and 5.0. The following table describes the differences between these versions.

Table 1-27 Differences between Redis versions

Feature	Redis 3.0	Redis 4.0 and Redis 5.0
Instance deployment mode	Based on VMs	Containerized based on physical servers
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds
QPS	100,000 QPS per node	100,000 QPS per node
Instance type	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby, and Redis Cluster
Instance total memory	Ranges from 2 GB to 1 TB.	Regular specifications range from 2 GB to 1 TB. Small specifications of 128 MB, 256 MB, 512 MB, and 1 GB are also available for single-node and master/standby instances.
Scale-up or 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

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.

- **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 1-28 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.	● Creates a Redis 3.0 instance in 5 to 15 minutes. ● Creates a containerized Redis 4.0 or 5.0 instance within 8 seconds.
Version	-	Closely follows open-source trends and supports the latest Redis version. Currently, Redis 3.0, 4.0, and 5.0 are supported.
Security	Network and server safety is the user's responsibility.	● Network security is ensured using VPCs and security groups. ● Data reliability is ensured by data replication and scheduled backup.
Performance	-	100,000 QPS per node

Feature	Open-Source Redis	DCS for Redis
Monitoring	Provides only basic statistics.	Provides more than 30 monitoring metrics and customizable alarm threshold and policies. ● Various metrics – External metrics include the number of commands, concurrent operations, connections, clients, and denied connections. – Resource usage metrics include CPU usage, physical memory usage, network input throughput, and network output throughput. – Internal metrics include instance capacity usage, as well as the number of keys, expired keys, PubSub channels, PubSub patterns, keyspace hits, and keyspace misses. ● Custom alarm thresholds and policies for different metrics to help identify service faults.
Backup and restoration	Supported	● Supports scheduled and manual backup. Backup files can be downloaded. ● Backup data can be restored on the console.
Parameter management	No visualized parameter management	● Visualized parameter management is supported on the console. ● Configuration parameters can be modified online.
Scale-up	Interrupts services and involves a complex procedure from modifying the server RAM to modifying Redis memory and restarting the OS and services.	● Supports online scale-up and scale-down without interrupting services. ● Specifications can be scaled up or down within the available range based on service requirements.

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.

Replica

A replica is a node of a DCS instance. No replication indicates that the instance does not have a standby node. Master/Standby replication indicates that the instance has a standby node. For example, a master/standby DCS instance has a master/standby replication. Each node of a cluster DCS Redis instance has a master/standby replication.

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 after the instance is created.

For details, see: **Modifying an Instance's Maintenance Time Window**.

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 standby node.

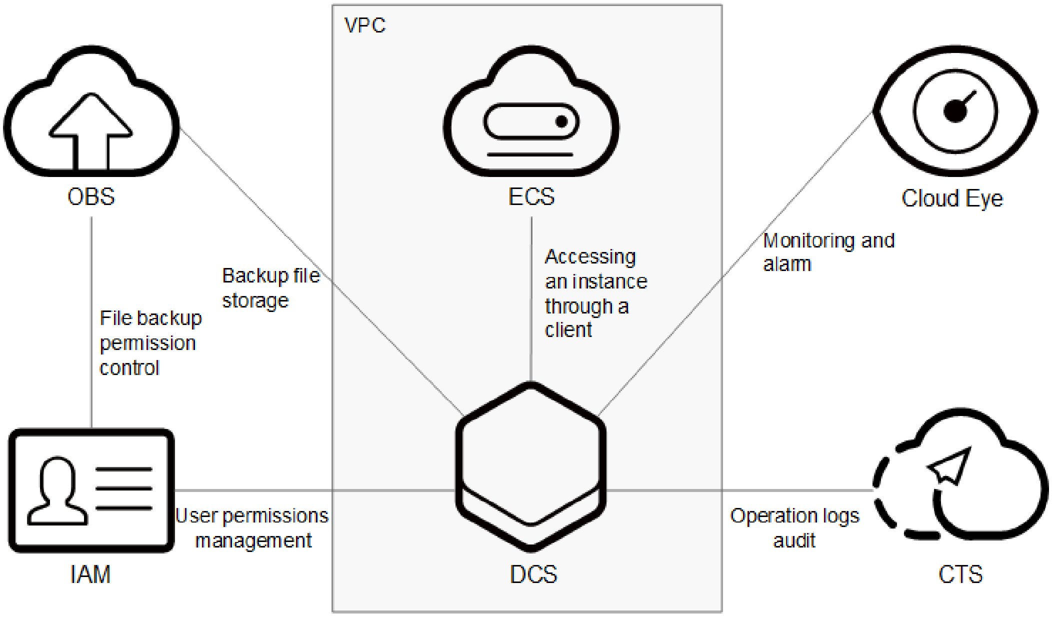
Shard

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.

1.10 Related Services

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

Figure 1-11 Relationships between DCS and other services


VPC
OBS
ECS
Cloud Eye
Backup file storage
File backup permission control
Accessing an instance through a client
Monitoring and alarm
User permissions management
Operation logs audit
IAM
DCS
CTS

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.

Cloud Trace Service (CTS)

CTS provides you with a history of operations performed on cloud service resources. With CTS, you can query, audit, and backtrack operations. The traces include the operation requests sent using the management console or open APIs and the results of these requests.

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 DCS 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 on the region and whether cross-AZ deployment is required.

   Choose a region closest to your application to reduce latency.

   A region consists of multiple availability zones (AZs) with physically isolated power supplies and networks. Master/standby and cluster DCS instances can be deployed across AZs.

   NOTE

   - If a master/standby or cluster DCS instance is deployed across AZs, faults in an AZ do not affect cache nodes in other AZs. This is because when the master node is faulty, the standby cache node will automatically become the master node to provide services. Such deployment achieves better disaster recovery.
   - Deploying a DCS instance across AZs slightly reduces network efficiency compared with deploying an instance within an AZ. Therefore, if a DCS instance is deployed across AZs, synchronization between master and standby cache nodes is slightly less efficient.

5. Decide whether backup policies are required.

   Currently, backup policies can be configured only for master/standby and cluster DCS instances. For details about backup and restoration, see **Overview**.

2.1.2 Preparing the Environment

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 a VPC, you can use it for all DCS instances you subsequently create.

Creating a VPC

Step 1 Log in to the management console.

Step 2 Click 📍 in the upper left corner of the management console 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 On the **Dashboard** page, click **Create VPC**.

Step 5 Create a VPC as prompted, retaining the default values unless otherwise required. For more information on how to create a VPC, see the *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 **6**. Otherwise, go to **7**.

Step 6 In the navigation pane on the left, choose **Subnet** and then click **Create Subnet** in the upper right corner of the displayed page. Create a subnet as prompted, retaining the default values unless otherwise required.

For more information on how to create a subnet, see the *Virtual Private Cloud User Guide*.

Step 7 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 more information on how to create a security group, see the *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.

Creating 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 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, 3.0, 4.0, and 5.0 versions are supported.

   NOTE

   - To create a Proxy Cluster instance, you can only select version 3.0.
   - To create a Redis Cluster instance, you can select versions 4.0 or 5.0.

3. Set **Instance Type** to **Single-node**, **Master/Standby**, **Proxy Cluster** or **Redis Cluster**.
4. Set **CPU Architecture** to **x86**.
5. Set **Replicas**. The default value is **2**.

   This parameter is displayed only when you select Redis 4.0 or Redis 5.0 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 sold out in an AZ, a sold-out tip will be displayed for the AZ. In this case, select another AZ.

   If the instance type is master/standby, Proxy Cluster, or Redis Cluster, **Standby AZ** is displayed. Select a standby AZ for the standby node of the instance.

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. For **VPC**, select a created VPC, subnet, and specify the IP address.

   You can choose to obtain an automatically assigned IP address or manually specify an IP address that is available in the selected subnet.

   For a DCS Redis 4.0 or 5.0 instance, you can specify a port numbering 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.

2. 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 can be configured only for instances that use Redis 3.0. DCS for Redis 4.0 and 5.0 are based on VPC endpoints and do not support security groups.

Step 7 Set the instance password.

This password is used for accessing the DCS Redis instance.

NOTE

For security purposes, you must enter an instance-specific password when you are accessing the DCS Redis instance. Keep your instance password secure and change it periodically.

Step 8 Click **More Settings** to display more configurations, including backup policy.

1. Specify **Name** and **Description**.

   The value of **Name** must be a string consisting of 4 to 64 characters.

2. Specify backup and restoration policies.

   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**.

3. Rename critical commands.

   Command Renaming is displayed for Redis 4.0 and 5.0. Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL** commands.

4. 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 9 Click **Create Now**.

The displayed page shows the instance information you have specified.

Step 10 Confirm the instance information and click **Submit**.

Step 11 Return to the **Cache Manager** page to view and manage your DCS instances.

1. Creating a single-node or master/standby DCS standard Redis instance takes 5 to 15 minutes. Creating a cluster DCS standard Redis instance takes 30 minutes.
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 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 5.0, 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 to serve as your Redis client. 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.

Procedure (Linux)

Step 1 Obtain the IP address and port number of the DCS Redis instance to be accessed.

For details, see **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 **http://download.redis.io/releases/redis-3.0.7.tar.gz**:

   wget http://download.redis.io/releases/redis-3.0.7.tar.gz
3. Run the following command to decompress the **redis-3.0.7** directory from the package:

   tar -xzf redis-3.0.7.tar.gz
4. Run the following commands to go to the **redis-3.0.7** directory and compile the source code of your Redis client:

   cd redis-3.0.7

   make

   cd src

Step 3 Access a DCS Redis instance which is not the Redis Cluster type.

Perform the following procedure to access a DCS Redis 3.0 instance, or a single-node or master/standby DCS Redis 4.0 or 5.0 instance.

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

   ./redis-cli -h {dcs_instance_address} -p 6379

   {dcs_instance_address} indicates the IP address of the DCS instance and **6379** is the port used for accessing the instance. The IP address and port number are obtained in **Step 1**.

   Example:

   ```
   [root@ecs-redis redis-3.0.7]# cd src
   [root@ecs-redis src]# ./redis-cli -h 192.168.0.148 -p 6379
   192.168.0.148:6379>
   ```

2. Enter the password. You can read and write cached data only after the password is verified.

   auth <password>

   <password> indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

   The command output is as follows:

   ```
   192.168.0.148:6379> auth *******
   OK
   192.168.0.148:6379>
   ```

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

Perform the following procedure to access a DCS Redis 4.0 or 5.0 instance in Redis Cluster type.

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 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 and port number are obtained in **Step 1**.

   Example:

   ```
   root@ecs-redis:~/redis-3.0.7/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
   0988ae8fd3686074c9afdcce73d7878c81a33ddc 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 Windows Redis client installation package. Decompress the package, open the CLI tool **cmd.exe**, and go to the directory where the decompressed Redis client installation package is saved. Then, run the following command to access the DCS Redis instance:

redis-cli -h *XXX* -p *6379*

XXX indicates the IP address of the DCS instance and **6379** is an example port number used for accessing a DCS instance. For details on how to obtain the IP address and port, see **Viewing Details of a DCS Instance**. Change the IP address and port as required.

2.2.2 Accessing a DCS Redis Instance Through Jedis

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

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.

Procedure

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

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS where you want to install Docker.

Step 3 Access the DCS instance by using Jedis.

1. Obtain the source code of the Jedis client from **https://github.com/xetorthio/jedis**.
2. Write code.

   Use either of the following two methods to access a DCS Redis instance through Jedis:

   - Single Jedis connection
   - Jedis pool

   Example code:

   a. Example code for a single Jedis connection

   ```
   // Creating a connection in password mode
    String host = "192.168.0.150";
    int port = 6379;
    String pwd = "passwd";

    Jedis client = new Jedis(host, port);
    client.auth(pwd);
    client.connect();
   // Run the set command
    String result = client.set("key-string", "Hello, Redis!");
   System.out.println( String.format("set instruction execution result:%s", result) );
   // Run the get command
    String value = client.get("key-string");
    System.out.println( String.format("get command result:%s", value) );

   // Creating a connection in password-free mode
    String host = "192.168.0.150";
    int port = 6379;

    Jedis client = new Jedis(host, port);
    client.connect();
   // Run the set command
    String result = client.set("key-string", "Hello, Redis!");
       System.out.println( String.format("set command result:%s", result) );
   // Run the get command
    String value = client.get("key-string");
    System.out.println( String.format("get command result:%s", value) );
   ```

   host indicates the example IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details on how to obtain the IP address/domain name and port, see **Step 1**. Change the IP address and port as required. **pwd** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. Do not hard code the plaintext password.

   b. Example code for a Jedis pool

   ```
   // Generate configuration information of a Jedis pool
    String ip = "192.168.0.150";
    int port = 6379;
    String pwd = "passwd";
    GenericObjectPoolConfig config = new GenericObjectPoolConfig();
    config.setTestOnBorrow(false);
    config.setTestOnReturn(false);
    config.setMaxTotal(100);
    config.setMaxIdle(100);
    config.setMaxWaitMillis(2000);
   JedisPool pool = new JedisPool(config, ip, port, 100000, pwd);//Generate a Jedis pool when the application is being initialized
   // Get a Jedis connection from the Jedis pool when a service operation occurs
   ```

```
 Jedis client = pool.getResource();
 try {
     // Run commands
     String result = client.set("key-string", "Hello, Redis!");
     System.out.println( String.format("set command result:%s", result) );
     String value = client.get("key-string");
     System.out.println( String.format("get command result:%s", value) );
 } catch (Exception e) {
     // TODO: handle exception
 } finally {
     // Return the Jedis connection to the Jedis connection pool after the client's request is
processed
     if (null != client) {
         pool.returnResource(client);
     }
 } // end of try block
 // Destroy the Jedis pool when the application is closed
 pool.destroy();

// Configure the connection pool in the password-free mode
 String ip = "192.168.0.150";
 int port = 6379;
 GenericObjectPoolConfig config = new GenericObjectPoolConfig();
 config.setTestOnBorrow(false);
 config.setTestOnReturn(false);
 config.setMaxTotal(100);
 config.setMaxIdle(100);
 config.setMaxWaitMillis(2000);
 JedisPool pool = new JedisPool(config, ip, port, 100000);//Generate a JedisPool when the
application is being initialized
// Get a Jedis connection from the Jedis pool when a service operation occurs
 Jedis client = pool.getResource();
 try {
     // Run commands
     String result = client.set("key-string", "Hello, Redis!");
     System.out.println( String.format("set command result:%s", result) );
     String value = client.get("key-string");
     System.out.println( String.format("get command result:%s", value) );
 } catch (Exception e) {
     // TODO: handle exception
 } finally {
     // Return the Jedis connection to the Jedis connection pool after the client's request is
processed
     if (null != client) {
         pool.returnResource(client);
     }
 } // end of try block
 // Destroy the Jedis pool when the application is closed
 pool.destroy();
```

ip indicates the IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details on how to obtain the IP address/domain name and port, see **Step 1**. Change the IP address and port as required. **pwd** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. Do not hard code the plaintext password.

3. Compile code according to the **readme** file in the source code of the Jedis client. Run the Jedis client to access the chosen DCS Redis instance.

----End

2.2.3 Accessing a DCS Redis 4.0 or 5.0 Instance on the Console

Access a DCS Redis instance through Web CLI. This function is supported only by DCS Redis 4.0 and 5.0 instances, and not by DCS Redis 3.0 instances.

NOTE

Do not enter sensitive information in Web CLI to avoid disclosure.

Prerequisites

The DCS Redis 4.0 or 5.0 instance you want to access through Web CLI is in the **Running** state.

Procedure

Step 1 Log in to the DCS console.

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

Step 3 In the navigation pane, choose **Cache Manager**. In the **Operation** column of the instance, choose **More** > **Connect to Redis**.

Step 4 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**.

----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 management 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.

  Enter a keyword to search.

- Select attributes and enter their keywords to search.

  Currently, you can search by name, ID, connection address (IP address:port number), AZ, status, instance type, cache engine, and CPU and memory type.

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. **Table 2-1** describes the parameters.

Table 2-1 Parameters on the Basic Information page of a DCS instance

Section	Parameter	Description
Instance Details	Name	Name of the chosen instance. To modify the instance name, click the ✎ icon.
	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 3.0.
	Instance Type	Type of the selected instance. Currently, supported types include single-node, master/standby, Proxy Cluster, and Redis Cluster.
	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: ● Size of data stored on the DCS instance ● Size of Redis-server buffers (including client buffer and repl-backlog) and internal data structures
	CPU | Memory Type	CPU and memory architecture of the chosen instance. This parameter is displayed only for DCS Redis instances.
	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 chosen DCS instance. To modify the description, click the ✎ icon.
Connection	Password Protected	Currently, only the password-protected mode is supported.
	Connection 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.
	Subnet	Subnet in which the chosen instance resides.

Section	Parameter	Description
	Security Group	Security group that controls access to the chosen instance. To modify the security group, click the ✎ icon. This parameter is displayed only for DCS Redis 3.0 instances. DCS for Redis 4.0 and 5.0 are based on VPC endpoints and do not support security groups.
Instance Topology	-	Hover over a node to view its metrics, or click the icon of a node to view its historical metrics. Topologies are supported only for master/standby and cluster instances.

----End

3 Operation Guide

3.1 Operating DCS Instances

3.1.1 Modifying DCS Instance Specifications

On the DCS console, you can modify a DCS Redis instance to the desired specification by expanding or reducing its capacity.

NOTE

- **Modify instance specifications during off-peak hours.**
- If your DCS instances are too old to support capacity expansion or reduction, contact technical support to upgrade the instances.

Note About Capacity Expansion or Reduction

- **The following table lists capacity expansion and reduction options supported by different DCS instances.**

Table 3-1 Capacity expansion and reduction options supported by different DCS instances

Cache Engine	Single-Node	Master/Standby	Cluster
Redis 3.0	Capacity expansion and reduction	Capacity expansion and reduction	Capacity expansion
Redis 4.0	Capacity expansion and reduction	Capacity expansion and reduction	Capacity expansion
Redis 5.0	Capacity expansion and reduction	Capacity expansion and reduction	Capacity expansion

- **Impact of capacity expansion and reduction:**
  - Single-node and master/standby

    The instance cannot be connected for several second and remains read-only for about 1 minute.

    For capacity expansion, only the memory of the instance is expanded. The CPU processing capability is not improved.

    Data of single-node instances may not be retained because they do not support data persistence. After the scaling, check whether the data is complete and import data if required.
  - Proxy Cluster

    The instance can be connected, but the CPU will be occupied and the latency will increase during data migration. During capacity expansion, new Redis Server nodes are added, and data is automatically balanced to the new nodes.

    Backup records created before the capacity change cannot be restored.
  - Redis Cluster

    The instance can be connected, but the CPU usage will increase and the latency will increase during data migration. During capacity expansion, new Redis Server nodes are added, and data is automatically balanced to the new nodes.

Procedure

Step 1 Log in to the DCS console.

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

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Choose **More** > **Modify Specifications** in the same row as the DCS instance.

Step 5 On the **Modify Specification** page, select the desired specification.

Step 6 Click **Submit**.

On the displayed **Background Tasks** page, 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.

----End

3.1.2 Restarting DCS Instances

On the DCS console, you can restart one or multiple DCS instances at a time.

⚠ WARNING

- 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.

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 of the management console 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 1 to 30 minutes to restart DCS instances. After DCS instances are restarted, their status changes to **Running**.

NOTE

- By default, only the instance process will restart. However, selecting **Force restart** will restart the VM on which the chosen DCS instance runs.
- To restart a single instance, you can also click **Restart** in the same row as the instance.

----End

3.1.3 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.
- 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 of the management console 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** or **Starting** state cannot be deleted.

Step 5 Choose **More** > **Delete** above the instance list.

Step 6 In the displayed dialog box, click **Yes**.

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 of the management console and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

If there are DCS instances that have failed to be created, **Instance Creation Failures** is displayed above the instance list.

Step 4 Click the icon or the number of failed tasks next to **Instance Creation Failures**.

The **Instance Creation Failures** dialog box is displayed.

Step 5 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

3.1.4 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

- During a master/standby node switchover, services will be interrupted for up to 10 seconds. Before performing this operation, ensure that your application supports connection re-establishment in case of disconnection.
- 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 of the management console 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**.

----End

3.1.5 Clearing DCS Instance Data

On the DCS console, you can clear data only for DCS Redis 4.0 and 5.0 instances.

Clearing instance data cannot be undone and cleared data cannot be recovered. Exercise caution when performing this operation.

Procedure

Step 1 Log in to the DCS console.

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

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Select one or more DCS2.0 instances to clear.

Step 5 Choose **More** > **Clear data** above the instance list.

Step 6 In the displayed dialog box, click **Yes**.

----End

3.1.6 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 of the management console and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Filter DCS instances to find the desired DCS instance.

Step 5 Click ↗ above the instance list.

Click the export result displayed in the lower left corner of the page.

----End

3.1.7 Command Renaming

After creating a DCS Redis 4.0 or 5.0 instance, you can rename the following critical commands: **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL**.

Procedure

Step 1 Log in to the DCS console.

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

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the **Operation** column of an instance, choose **More** > **Command Renaming**.

Step 5 Select a command, enter a new name, and click **OK**.

📖 **NOTE**

- You can rename multiple commands at a time.
- The new command names will take effect only after you restart the instance. Remember the new command names because they will not be displayed on the console for security purposes.

----End

3.2 Managing DCS Instances

3.2.1 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 the administrator or perform these operations after the cache nodes recover from faults.

3.2.2 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**.

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 of the management console 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 On the instance details page, click the **Parameters** tab.

Step 6 On the **Parameters** tab page, click **Modify**.

Step 7 Modify parameters based on your requirements.

Table 3-2 describes the parameters. In most cases, retain the default values.

Table 3-2 DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
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 **0** means that this function is disabled.	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.	● no ● always ● everysec	everysec
appendonly	Indicates whether to log each modification of the instance. 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	● yes ● no	yes

Parameter	Description	Value Range	Default Value
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above **client-output-buffer-slave-soft-limit** before the client is disconnected.	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.	0–17,179,869,184	1,717,986,918
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 **client-output-buffer-limit-slave-soft-seconds** parameter, the client is disconnected.	0–17,179,869,184	1,717,986,918
maxmemory-policy	The policy applied when the maxmemory limit is reached. For more information about this parameter, see **https://redis.io/topics/lru-cache**.	volatile-lru allkeys-lru volatile-random allkeys-random volatile-ttl noeviction	noeviction
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	5000	100–5000
master-read-only	Sets the instance to be read-only. All write operations will fail.	● yes ● no	no
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance.	1000–10,000	10,000

Parameter	Description	Value Range	Default Value
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.	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 **0** indicates that the backlog is never released.	0–604,800	3,600
repl-timeout	Replication timeout (in seconds).	30–3,600	60
hash-max-ziplist-entries	The maximum number of hashes that can be encoded using ziplist, a data structure optimized to reduce memory use.	1–10,000	512
hash-max-ziplist-value	The largest value allowed for a hash encoded using ziplist, a special data structure optimized for memory use.	1–10,000	64

Parameter	Description	Value Range	Default Value
set-max-intset-entries	If a set is composed entirely of strings that are integers in radix 10 within the range of 64 bit signed integers, the set is encoded using intset, a data structure optimized for memory use.	1–10,000	512
zset-max-ziplist-entries	The maximum number of sorted sets that can be encoded using ziplist, a data structure optimized to reduce memory use.	1–10,000	128
zset-max-ziplist-value	The largest value allowed for a sorted set encoded using ziplist, a special data structure optimized for memory use.	1–10,000	64
latency-monitor-threshold	The minimum amount of latency that will be logged as latency spikes If this parameter is set to **0**, latency monitoring is disabled. If this parameter is set to a value greater than 0, all events blocking the server for a time greater than the configured value will be logged. By running the **LATENCY** command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring. For more information about the latency-monitor-threshold, visit **https://redis.io/topics/latency-monitor**.	0–86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	Controls which keyspace events notifications are enabled for. If this parameter is configured, the Redis Pub/Sub feature will allow clients to receive an event notification when a Redis data set is modified.	A string of different values can be used to enable notifications for multiple event types: Possible values include: K: Keyspace events, published with the __keyspace@__ prefix E: Keyevent events, published with __keyevent@__ prefix g: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME $: String commands l: List commands s: Set commands h: Hash commands z: Sorted set commands x: Expired events (events generated every time a key expires) e: Evicted events (events generated when a key is evicted from maxmemory) For more information, see the following note.	Ex
slowlog-log-slower-than	The maximum amount of time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis Slow Log will record the command.	0–1,000,000	10,000
slowlog-max-len	The maximum allowed length of the Redis Slow Log logs. Slow Log consumes memory, but you can reclaim this memory by running the SLOWLOG RESET command.	0–1000	128

NOTE

1. For more information about the parameters described in **Table 3-2**, visit **https://redis.io/topics/memory-optimization**.
2. 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.
3. 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 **Kl** 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.

Step 8 After you have finished setting the parameters, click **Save**.

Step 9 Click **Yes** to confirm the modification.

----End

3.2.3 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 of the management console 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

3.2.4 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 or 5.0 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 of the management console 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**.

The modification will take effect immediately, that is, the new maintenance time window will appear on the **Basic Information** tab page immediately.

----End

3.2.5 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 of the management console 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 whose background task you want to manage.

Step 5 Click the **Background Tasks** tab.

A list of background tasks is displayed.

Step 6 Click 📅 , specify **Start Date** and **End Date**, and click **OK** to view tasks started in the corresponding time segment.

- 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

3.2.6 Viewing Data Storage Statistics of a Proxy Cluster Instance

You can view the data storage statistics of all nodes of a 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 Proxy Cluster instances. Instances of other types only have one node, and you can view the used memory on the instance details page.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management 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 cluster instance to view the basic information.

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

3.2.7 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 and 80 for Lists/Sets/Zsets/Hashes) 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:

- Only DCS Redis 4.0 and 5.0 instances support hot key analysis, 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 of the management 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 Click the **Cache Analysis** tab.

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.

----End

Hot Key Analysis Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management 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 Click the **Cache Analysis** tab.

Step 6 On the **Hot Key Analysis** tab page, manually perform hot key analysis or schedule daily automatic analysis.

NOTE

The default value of the **maxmemory-policy** parameter of a Redis 4.0 or 5.0 instance is **noeviction**. To perform hot key analysis, set this parameter 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 3-3 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. **FREQ** is the logarithmic access frequency counter. The maximum value of **FREQ** is 255, which indicates 1 million access requests. After **FREQ** reaches **255**, it will no longer increment even if access requests continue to increase. **FREQ** will decrement by 1 for every minute during which the key is not accessed.

Parameter	Description
DataBase	Database where a hot key is located.

----End

3.2.8 Viewing Redis Slow Logs

Redis uses slow logs to record queries that exceed a specified execution time. You can view the slow logs on the DCS console to identify performance issues.

For details about the commands, visit the **Redis official website**.

Configure the slow log with the following parameters:

- **slowlog-log-slower-than**: The maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis slow log will record the command. The default value is **10,000**. That is, if command execution exceeds 10 ms, the command will be recorded in the slow log.
- **slowlog-max-len**: The maximum allowed length of the slow log. The default value is **128**. That is, if the number of records in the slow log exceeds 128, the earliest record will be deleted to make room for new ones.

For details about the configuration parameters, see **Modifying Configuration Parameters**.

Viewing Slow Logs on the Console

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management 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 instance.

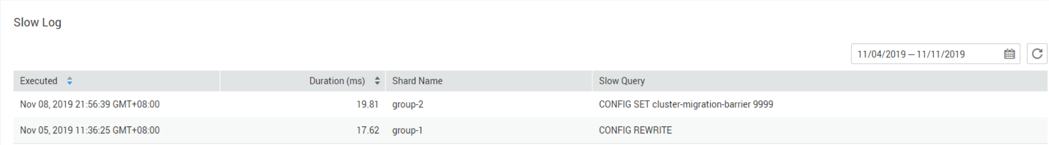
Step 5 Click the **Slow Log** tab.

Step 6 Select a start date and an end date to view slow log records within the specified period.

NOTE

For details about the commands, visit the **Redis official website**.

Figure 3-1 Slow log records of an instance



Slow Log

11/04/2019 – 11/11/2019

Executed	Duration (ms)	Shard Name	Slow Query
Nov 08, 2019 21:56:39 GMT+08:00	19.81	group-2	CONFIG SET cluster-migration-barrier 9999
Nov 05, 2019 11:36:25 GMT+08:00	17.62	group-1	CONFIG REWRITE

----End

3.2.9 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.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management 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 instance.

Step 5 Click the **Run Logs** tab.

Step 6 Click **Create Log File**.

If the instance is the master/standby or cluster type, you can specify the shard and replica whose run logs you want to collect. If the instance is the single-node type, logs of the only node of the instance will be collected.

----End

3.3 Backing Up and Restoring DCS Instances

3.3.1 Overview

On the DCS console, you can back up and restore DCS instances.

Importance of DCS Instance Backup

There is a small chance that dirty 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.

Backup Modes

DCS instances support the following backup modes:

- Scheduled 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 also be issued manually. Then, data in the chosen DCS instances will be permanently backed up to OBS. Backup data can be deleted manually.

  Before performing high-risk operations, such as system maintenance or upgrade, back up DCS instance data.

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

  Instance data is persisted using the Redis Append Only Files (AOF) feature.

  Backup tasks are 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 are 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 and pricing 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.

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 are 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.

3.3.2 Configuring a 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.

If automatic backup is not required, disable the automatic backup function in the backup policy.

Prerequisites

At least one master/standby DCS instance has been created.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console 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 the ○━ to the right to enable automatic backup. Backup policies will be displayed.

Table 3-4 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. **NOTE** 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 **Running** state can be backed up.

Step 7 Click **OK**.

----End

3.3.3 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 of the management console 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 In the **Create Backup** dialog box, click **OK**.

Information in the **Description** text box cannot exceed 128 bytes.

> **NOTE**
>
> Instance backup takes 10 to 15 minutes. The data added or modified during the backup process will not be backed up.

----End

3.3.4 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 **succeed**.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console 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 5 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.

----End

3.3.5 Downloading a Backup File

Due to the limitations of automatic and manual backups (automatically backed up data can be retained for a maximum of 7 days, and manually backed up data takes space in OBS), you should download the backup files and permanently save them on the local host.

This function is supported only by master/standby instances, and not by single-node instances.

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 of the management console 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 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

3.4 Migrating Data with DCS

3.4.1 Introduction to Migration with DCS

DCS for Redis provides the following migration modes:

- Importing data from backup files: Download the source Redis data and then upload the data to an OBS bucket in the same region as the destination DCS Redis instance. DCS will read the backup data from the OBS bucket and migrate the data into the destination instance.

  This migration mode can be used for migrating data from other Redis vendors or self-hosted Redis to DCS for Redis.
- Migrating data online: If the source and destination 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 destination.

The following table describes data migration modes supported by DCS.

Table 3-5 DCS data migration modes

Migration Mode	Source	Destination: DCS		
		Single-Node and Master/Standby	Proxy Cluster	Redis Cluster
Importing backup files	OBS bucket: AOF files	√	√	×
	OBS bucket: RDB files	√	√	√
Migrating data online	DCS for Redis: single-node or master/standby	√	√	√
	DCS for Redis: Proxy Cluster	×	×	×
	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	×	×	×

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.

3.4.2 Importing Backup Files

Application Scenarios

Use the DCS console to migrate Redis data from Redis of other vendors or self-hosted Redis to DCS for Redis.

Simply download the source Redis data and then upload the data to an OBS bucket in the same region as the destination 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 destination 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 destination DCS Redis instance.
- The data files to be uploaded must be in the .aof, .rdb, .zip, or .tar.gz format.
- 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.
- .rdb backup files of self-hosted Redis 5.0 cannot be imported. .rdb backup files of self-hosted Redis 3.0 or 4.0 can be exported using redis-cli. .rdb files of other cloud Redis can be exported only by creating backup tasks, and cannot be exported by running commands in redis-cli.
- Redis Cluster instances only support .rdb files and do not support .aof files.

Step 1: Prepare the Destination 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 must clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

You can use a DCS Redis 3.0, 4.0, or 5.0 instance as the destination instance.

Step 2: Create an OBS Bucket and Upload 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 destination DCS Redis instance.
3. Specify **Bucket Name**.

   The bucket name must meet the naming rules specified on the console.
4. Set **Storage Class** to **Standard** or **Infrequent Access**.

   Do not select **Archive**. Otherwise, the migration will fail.
5. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
6. Specify **Multiple AZ Mode** and **Tags** as instructed.
7. Click **Create Now**.

Step 2 Upload the backup data files to the OBS bucket.

If the backup file to be uploaded is smaller than 50 MB, go to step **Step 3** to upload the file using the OBS console.

If the backup file to be uploaded is larger than 50 MB, perform the following steps to upload the file using OBS Browser.

1. Configure user permissions.

For details, see the *OBS Console Operation Guide* > *Getting Started* > *Configuring User Permissions*.

2. Download OBS Browser.

   For details, see the *OBS Browser Operation Guide* > *Getting Started* > *Downloading OBS Browser*.

3. Create access keys (AK and SK).

   For details, see the *OBS Browser Operation Guide* > *Getting Started* > *Creating Access Keys (AK and SK)*.

4. Log in to OBS Browser.

   For details, see the *OBS Browser Operation Guide* > *Getting Started* > *Logging In to OBS Browser*.

5. Add a bucket.

   For details, see the *OBS Browser Operation Guide* > *Getting Started* > *Adding a Bucket*.

6. Upload backup data.

   For details, see the *OBS Browser Operation Guide* > *Getting Started* > *Uploading a File or Folder*.

Step 3 On the OBS console, upload the backup data files to the OBS bucket.

The following steps are applicable if the backup files are smaller than 50 MB.

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. If **Upload Mode** is set to **Batch**, a maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.

   To upload objects, drag files or folders to the **Upload Object** area or click **add file**.

5. If **Upload Mode** is set to **Single**, upload a single file at a time. The maximum size of the file is 50 MB.

   Click ... to open the local file browser, select the file to be uploaded, and click **Open**.

6. Specify **Storage Class**.

   Do not select **Archive**. Otherwise, the migration will fail.

7. (Optional) Select **KMS encryption** to encrypt the file you want to upload.

   KMS encryption is not supported when a local backup file is uploaded to an OBS bucket.

8. Click **Upload**.

----End

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

Step 3 Click **Create Migration Task**.

Step 4 Specify **Task Name** and **Description**.

Step 5 Set **Migration Mode** to **Import backup file**.

Data Source is automatically set to **OBS bucket**.

Step 6 For **OBS Bucket**, select the OBS bucket to which you have uploaded backup files.

In the **Backup Files** table, the files you have uploaded are displayed.

NOTE

You can upload files in the .aof, .rdb, .zip, or .tar.gz format.

Step 7 Select the backup files whose data is to be migrated.

Step 8 Select the destination instance created in **Step 1: Prepare the Destination DCS Redis Instance**.

Step 9 Enter the password of the destination instance. Click **Test Connection** to verify the password.

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**.

----End

3.4.3 Migrating Data Online

Application Scenarios

If the source and destination 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 destination.

Prerequisites

Before migrating data, read through **Introduction to Migration with DCS** to learn about the DCS data migration function and select an appropriate destination instance.

Step 1: Obtain the Source Instance Address

Step 1 Log in to the DCS console.

Step 2 Confirm the type and version of the source instance.

Step 3 On the details page of the source instance, obtain the connection address (IP address:port) of instance.

----End

Step 2: Prepare the Destination DCS Redis Instance

- If a destination DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If a destination 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**.

Step 3: Check the Network Between the Source and Destination Instances

Step 1 Check whether the source and destination instances are in the same VPC.

If yes, go to **Step 4: Create a Migration Task**. If no, go to **Step 2**.

Step 2 Check whether the VPCs of the source and destination instances are connected.

If yes, go to **Step 1**. If no, create a VPC peering connection. For details, see "VPC Peering Connection" section in the *VPC User Guide*.

----End

Step 4: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

Step 3 Click **Create Migration Task**.

Step 4 Specify **Task Name** and **Description**.

Step 5 Set **Migration Mode** to **Migrate data online**.

Step 6 Specify **Migration Type**.

Supported migration types are **Full** and **Full + incremental**, which are described in **Table 3-6**.

Table 3-6 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 destination instance.
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and destination instances. You can manually stop an ongoing migration.

Step 7 Enter the source instance address obtained in **Step 1: Obtain the Source Instance Address**.

Set **Source Redis Type** to either of the following:

- **Intra-VPC Redis**: The source Redis and the destination instance are in the same VPC.
- **Self-hosted Redis using Direct Connect or VPN connection**: The source Redis and the destination instance are interconnected through VPN connection or Direct Connect connection.

Step 8 For **Destination Instance**, select the DCS Redis Instance prepared in **Step 2: Prepare the Destination DCS Redis Instance**.

If the instance is password-protected, you can click **Test Connection** to check whether the instance password meets the requirements.

Step 9 Click **Next**.

Step 10 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**.

----End

3.5 Managing Passwords

3.5.1 DCS Instance Passwords

Passwords can be configured to control access to your DCS instances, ensuring the security of your data.

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.

3.5.2 Changing Instance Passwords

On the DCS console, you can change the password required for accessing your DCS instance.

NOTE

- The DCS instance for which you want to change the password is in the **Running** state.
- The new password will not take effect in the client until the client is restarted.

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 of the management console 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.

Step 6 In the **Change Password** dialog box, click **OK** to confirm the password change.

----End

4 Monitoring

4.1 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.

Different types of instances are monitored on different dimensions.

- Single-node:

  Single-node instances are monitored on the instance dimension. The monitoring is conducted on the Redis Server.

- Master/standby:

  Master/Standby instances are monitored on the instance and Redis Server dimensions. Instance monitoring covers the master node, while Redis Server monitoring covers the master and standby nodes.

- Cluster:

  Proxy Cluster instances are monitored on the instance, Redis Server, and proxy dimensions. Instance monitoring covers the aggregated master node data, Redis Server monitoring covers each shard in the cluster, and proxy monitoring covers each proxy in the cluster.

  Redis Cluster instances are monitored on the instance and Redis Server dimensions. Instance monitoring covers the aggregated master node data and Redis Server monitoring covers each shard in the cluster.

Namespace

SYS.DCS

DCS Redis 3.0 Instance Metrics

NOTE

The **Monitored Objects and Dimensions** column lists instances and dimensions that support the corresponding metrics.

Table 4-1 DCS Redis 3.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	CPU consumed by the monitored object Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
net_in_throughput	Network Input Throughput	Inbound throughput per second on a port Unit: byte/s	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
net_out_throughput	Network Output Throughput	Outbound throughput per second on a port Unit: byte/s	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	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	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between **used_memory_rss**/ **used_memory**.	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: kbit/s	≥ 0 kbits/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: kbit/s	≥ 0 kbits/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_hits	Keyspace Hits	Number of successful lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
auth_errors	Authentication Failures	Number of failed authentications	≥ 0	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance	- **1**: yes - **0**: no	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute
keys	Keys	Number of keys in Redis	≥ 0	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute

DCS Redis 4.0 and 5.0 Instance Metrics

NOTE

The **Monitored Objects and Dimensions** column lists instances and dimensions that support the corresponding metrics.

Table 4-2 DCS Redis 4.0 and 5.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	CPU consumed by the monitored object Unit: %	0–100%	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance	- **1**: yes - **0**: no	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expires	Keys With an Expiration	Number of keys with an expiration in Redis	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset_perc	Used Memory Dataset Ratio	Percentage of dataset memory that the Redis server has used Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
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	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
keys	Keys	Number of keys in Redis	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
del	DEL	Number of **DEL** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expire	EXPIRE	Number of **EXPIRE** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
get	GET	Number of **GET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/ standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hdel	HDEL	Number of **HDEL** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
hget	HGET	Number of **HGET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
hmget	HMGET	Number of **HMGET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
hmset	HMSET	Number of **HMSET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hset	HSET	Number of **HSET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Ratio between Used Memory RSS and Used Memory	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mget	MGET	Number of **MGET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
mset	MSET	Number of **MSET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
set	SET	Number of **SET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
sadd	Sadd	Number of **SADD** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
smembers	Smembers	Number of **SMEMBERS** commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period Unit: Count	≥ 0	Monitored object: Redis Cluster instance Dimension: dcs_instance_id	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Monitored object: Redis Cluster instance Dimension: dcs_instance_id	1 minute

Cluster DCS Redis Instance Metrics

NOTE

- The following describes the metrics for cluster DCS instances. For Proxy Cluster DCS Redis 3.0 instances, the monitoring covers Redis Servers and Proxies. For Redis Cluster DCS Redis 4.0 and 5.0 instances, the monitoring only covers Redis Servers. For details, see **Table 4-3** and **Table 4-4**.
- The **Monitored Objects and Dimensions** column lists instances and dimensions that support the corresponding metrics.

Table 4-3 Redis Server metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	CPU consumed by the monitored object Unit: %	0–100%	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
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	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between **used_memory_rss**/**used_memory**.	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups Unit: %	0–100%	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the node	- **1**: yes - **0**: no	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
keys	Keys	Number of keys in Redis	≥ 0	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
sadd	Sadd	Number of **SADD** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
smembers	Smembers	Number of **SMEMBERS** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
ms_repl_offset	Replication Gap	Data synchronization gap between the master and the replica	-	Monitored object: Replica of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
del	DEL	Number of **DEL** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
expire	EXPIRE	Number of **EXPIRE** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
get	GET	Number of **GET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
hdel	HDEL	Number of **HDEL** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
hget	HGET	Number of **HGET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hmget	HMGET	Number of **HMGET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
hmset	HMSET	Number of **HMSET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
hset	HSET	Number of **HSET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
mget	MGET	Number of **MGET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mset	MSET	Number of **MSET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
set	SET	Number of **SET** commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period Unit: Count	≥ 0	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Monitored object: Redis Server of cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute

Table 4-4 Proxy metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	CPU consumed by the monitored object Unit: %	0–100%	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	1 minute
p_connected_clients	Connected Clients	Number of connected clients	≥ 0	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	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	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	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	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	1 minute
max_rxkB_per_sec	Maximum Inbound Bandwidth	Largest volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	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	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	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	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
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	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	1 minute
avg_rxkB_per_sec	Average Inbound Bandwidth	Average volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	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	Monitored object: Proxy in a DCS Redis 3.0 instance Dimension: dcs_cluster_proxy_node	1 minute

Dimensions

Key	Value
dcs_instance_id	DCS Redis instance
dcs_cluster_redis_node	Redis Server
dcs_cluster_proxy_node	Proxy

4.2 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 of the management console 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

4.3 Configuring Alarm Rules for Critical Metrics

This section describes the alarm rules of some metrics and how to configure the rules. In actual scenarios, configure alarm rules for metrics by referring to the following alarm policies.

Alarm Policies for DCS Redis Instances

Table 4-5 DCS Redis instance metrics to configure alarm rules for

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
CPU Usage	0–100	Alarm threshold: 70 Number of consecutive periods: 2 Alarm severity: Major	No	Consider capacity expansion based on the service analysis. The CPU capacity of a single-node or master/standby instance cannot be expanded. If you need larger capacity, use a cluster instance instead.

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Memory Usage	0–100	Alarm threshold: 70 Number of consecutive periods: 2 Alarm severity: Major	No	Expand the capacity of the instance.
Connected Clients	0–10,000	Alarm threshold: 8000 Number of consecutive periods: 2 Alarm severity: Major	No	Optimize the connection pool in the service code to prevent the number of connections from exceeding the maximum limit. For single-node and master/standby instances, the maximum number of connections allowed is 10,000. You can adjust the threshold based on service requirements.
New Connections (Count/min)	0–10,000	Alarm threshold: 10,000 Number of consecutive periods: 2 Alarm severity: Minor	-	Check whether **connect** is used and whether the client connection is abnormal. Use persistent connections ("pconnect" in Redis terminology) to ensure performance.
Input Flow	> 0	Alarm threshold: 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. Configure this alarm only for single-node and master/standby DCS Redis 3.0 instances and set the alarm threshold to 80% of the assured bandwidth of DCS Redis 3.0 instances.

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Output Flow	> 0	Alarm threshold: 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. Configure this alarm only for single-node and master/standby DCS Redis 3.0 instances and set the alarm threshold to 80% of the assured bandwidth of DCS Redis 3.0 instances.

Procedure

In the following example, an alarm rule is set for the **CPU Usage** metric.

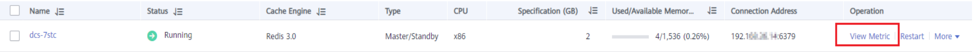
Step 1 Log in to the DCS console.

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

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the same row as the DCS instance whose metrics you want to view, choose **More** > **View Metric**.

Figure 4-1 Viewing instance metrics



□	Name	Status	Cache Engine	Type	CPU	Specification (GB)	Used/Available Memor...	Connection Address	Operation
□	dcs-7stc	Running	Redis 3.0	Master/Standby	x86	2	4/1,536 (0.26%)	192.1[redacted]:6379	View Metric Restart More ▾

Step 5 Locate the **CPU Usage** metric. Hover over the metric and click + to create an alarm rule for the metric.

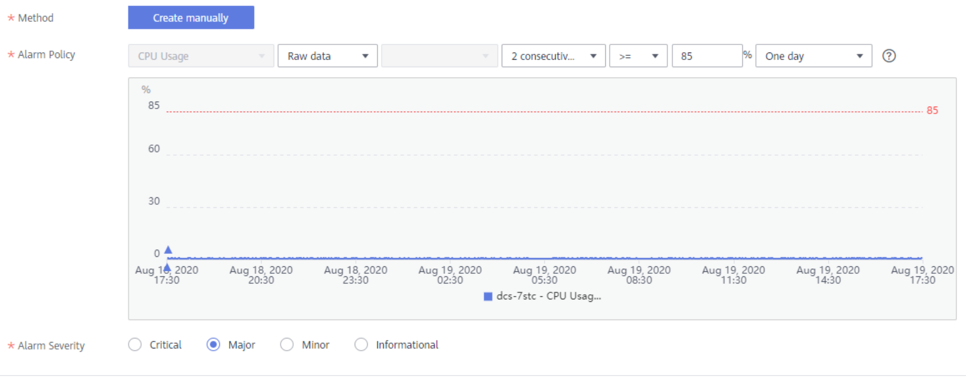
The **Create Alarm Rule** page is displayed.

Step 6 Specify the alarm rule details.

1. Specify the alarm policy and alarm severity.

   For example, the alarm policy shown in the following figure indicates that an alarm will be triggered if the number of connected clients exceeds the threshold for two consecutive periods.

Figure 4-2 Setting the alarm content



* Method: Create manually

* Alarm Policy: CPU Usage | Raw data | | 2 consecutiv... | >= | 85 % | One day

* Alarm Severity: ○ Critical ● Major ○ Minor ○ Informational

2. Set the alarm notification configurations. If you enable **Alarm Notification**, set the validity period, notification object, and trigger condition.
3. Click **Next**.
4. Under **Specify Rules Name**, set the alarm name and description.
5. Click **Create**.

NOTE

For more information about creating alarm rules, see the *Cloud Eye User Guide* > *Using the Alarm Function* > *Creating Alarm Rules*.

----End

5 Auditing

5.1 Operations That Can Be Recorded by CTS

With CTS, you can query, audit, and review operations performed on cloud resources. Traces include the operation requests sent using the management console or open APIs as well as the results of these requests.

The following lists the DCS operations that can be recorded by CTS.

Table 5-1 DCS operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Creating an instance	DCS	createDCSInstance
Submitting an instance creation request	DCS	submitCreateDCSInstanceRequest
Deleting multiple instances	DCS	batchDeleteDCSInstance
Deleting an instance	DCS	deleteDCSInstance
Modifying instance information	DCS	modifyDCSInstanceInfo
Modifying instance configurations	DCS	modifyDCSInstanceConfig

Operation	Resource Type	Trace Name
Changing instance password	DCS	modifyDCSInstancePassword
Restarting an instance	DCS	restartDCSInstance
Submitting an instance restarting request	DCS	submitRestartDCSInstanceRequest
Starting an instance	DCS	startDCSInstance
Submitting an instance starting request	DCS	submitStartDCSInstanceRequest
Clearing instance data	DCS	flushDCSInstance
Restarting instances in batches	DCS	batchRestartDCSInstance
Submitting a request to restart instances in batches	DCS	submitBatchRestartDCSInstanceRequest
Starting multiple instances	DCS	batchStartDCSInstance
Submitting a request to start instances in batches	DCS	submitBatchStartDCSInstanceRequest
Restoring instance data	DCS	restoreDCSInstance
Submitting a request to restore instance data	DCS	submitRestoreDCSInstanceRequest

Operation	Resource Type	Trace Name
Backing up instance data	DCS	backupDCSInstance
Submitting a request to back up instance data	DCS	submitBackupDCSInstanceRequest
Deleting instance backup files	DCS	deleteInstanceBackupFile
Deleting background tasks	DCS	deleteDCSInstanceJobRecord
Modifying instance specifications	DCS	modifySpecification
Submitting a request to modify instance specifications	DCS	submitModifySpecificationRequest
Creating an instance subscription order	DCS	createInstanceOrder
Switching between master and standby nodes	DCS	masterStandbySwitchover
Resetting instance password	DCS	resetDCSInstancePassword
Submitting a request to clear instance data	DCS	submitFlushDCSInstanceRequest

5.2 Viewing Traces on the CTS Console

After CTS is enabled, the tracker starts recording operations on cloud resources. Operation records for the last seven days can be viewed on the CTS console. This section describes how to query operation records of the last seven days on the CTS console.

Procedure

Step 1 Log in to the management console.

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

NOTE

Select the same region as your application service.

Step 3 Click **Service List** and choose **Management & Deployment** > **Cloud Trace Service**.

Step 4 In the navigation pane, click **Trace List**.

Step 5 Specify the filters used for querying traces. The following filters are available:

- **Search By**:

  Select an option from the drop-down list. Select **DCS** from the **Trace Source** drop-down list.

  When you select **Trace name**, you also need to select a specific trace name.

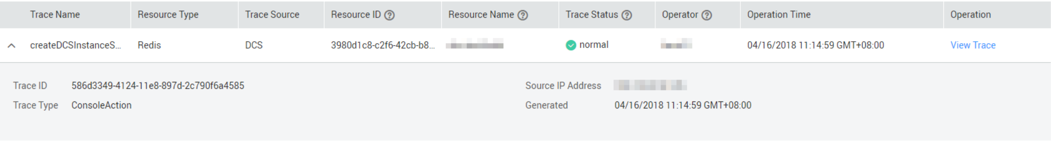
  When you select **Resource ID**, you also need to select a specific resource ID.

  When you select **Resource name**, you also need to select a specific resource name.

- **Operator**: Select a specific operator (a user other than tenant).
- **Trace Status**: Available options include **All trace status**, **normal**, **warning**, and **incident**. You can select only one of them.
- Start time and end time: You can specify the time period in which to query traces.

Step 6 Click ∨ on the left of a trace to expand its details, as shown in **Figure 5-1**.

Figure 5-1 Expanding trace details

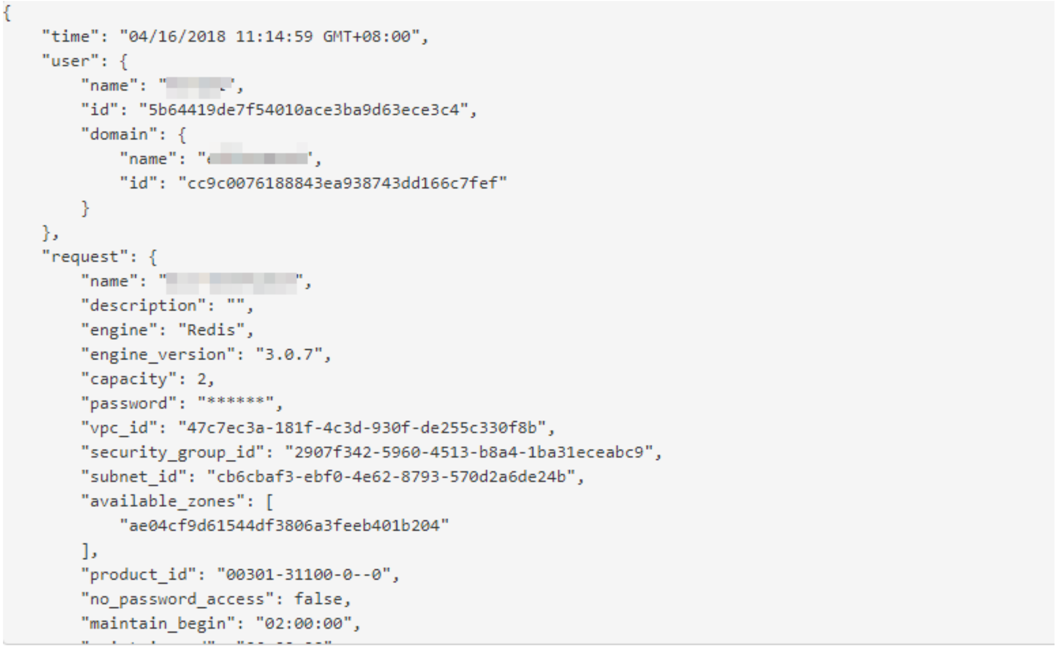


	Trace Name	Resource Type	Trace Source	Resource ID ⓘ	Resource Name ⓘ	Trace Status ⓘ	Operator ⓘ	Operation Time	Operation
∧	createDCSInstanceS...	Redis	DCS	3980d1c8-c2f6-42cb-b8...	[redacted]	✓ normal	[redacted]	04/16/2018 11:14:59 GMT+08:00	View Trace

Trace ID 586d3349-4124-11e8-897d-2c790f6a4585
Trace Type ConsoleAction
Source IP Address [redacted]
Generated 04/16/2018 11:14:59 GMT+08:00

Step 7 Click **View Trace** in the **Operation** column. In the dialog box shown in **Figure 5-2**, the trace structure details are displayed.

Figure 5-2 Viewing traces



```
{
    "time": "04/16/2018 11:14:59 GMT+08:00",
    "user": {
        "name": "[redacted]",
        "id": "5b64419de7f54010ace3ba9d63ece3c4",
        "domain": {
            "name": "[redacted]",
            "id": "cc9c0076188843ea938743dd166c7fef"
        }
    },
    "request": {
        "name": "[redacted]",
        "description": "",
        "engine": "Redis",
        "engine_version": "3.0.7",
        "capacity": 2,
        "password": "******",
        "vpc_id": "47c7ec3a-181f-4c3d-930f-de255c330f8b",
        "security_group_id": "2907f342-5960-4513-b8a4-1ba31eceabc9",
        "subnet_id": "cb6cbaf3-ebf0-4e62-8793-570d2a6de24b",
        "available_zones": [
            "ae04cf9d61544df3806a3feeb401b204"
        ],
        "product_id": "00301-31100-0--0",
        "no_password_access": false,
        "maintain_begin": "02:00:00",
```

----End

6 FAQs

6.1 Client and Network Connection

6.1.1 Security Group Configurations

This section describes how to configure a security group for accessing a DCS instance **within a VPC**.

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.

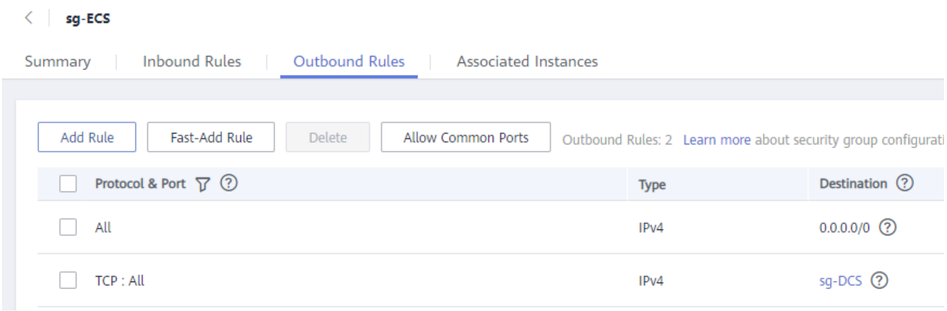
- If the ECS and DCS instance are configured with the same security group, network access in the group is not restricted by default.
- If the ECS and DCS instance are configured with different security groups, add security group rules to ensure that the ECS and DCS instance can access each other.

  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.

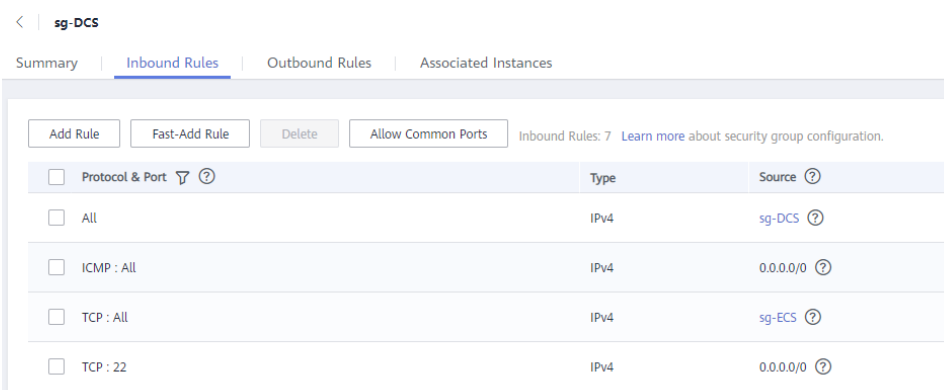


Outbound Rules: 2 Learn more about security group configurati

☐	Protocol & Port	Type	Destination
☐	All	IPv4	0.0.0.0/0
☐	TCP : All	IPv4	sg-DCS

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:



Inbound Rules: 7 Learn more about security group configuration.

☐	Protocol & Port	Type	Source
☐	All	IPv4	sg-DCS
☐	ICMP : All	IPv4	0.0.0.0/0
☐	TCP : All	IPv4	sg-ECS
☐	TCP : 22	IPv4	0.0.0.0/0

NOTICE

For the source IP address, use the specified IP address of the DCS instance. Avoid using **0.0.0.0/0** to prevent ECSs bound with the same security group from being attacked by Redis vulnerability exploits.

6.1.2 Does DCS Support Public Access?

No. 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.

In the application development and debugging phase, you can also use an SSH agent to access DCS instances in the local environment.

6.1.3 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.

6.1.4 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.

6.1.5 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 timeout errors occur frequently, contact O&M personnel.

6.1.6 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, 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 the administrator.

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

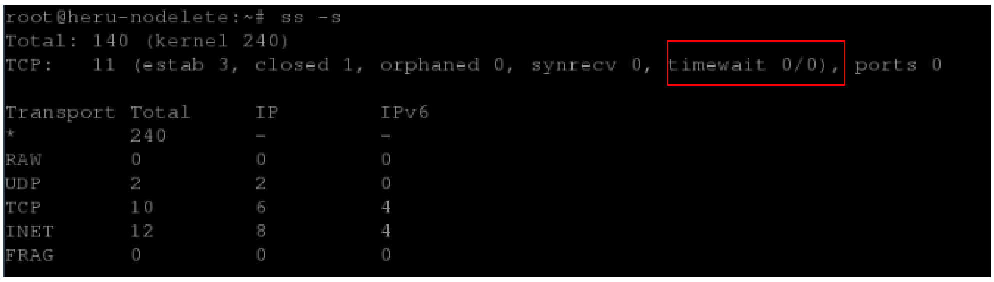
Step 3 Check the JedisPool code.

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 the administrator 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.

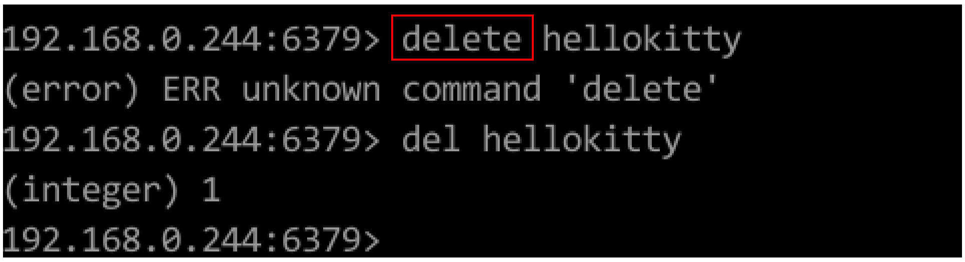
----End

6.1.7 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.

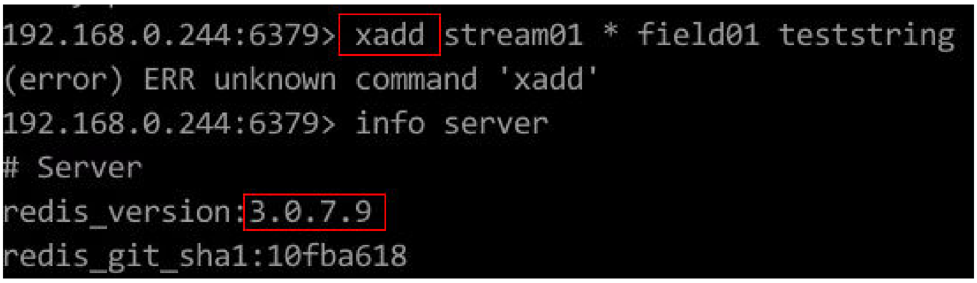
   As shown in the following figure, the error message is returned because the correct command for deleting a string should be **del**.



```
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**.

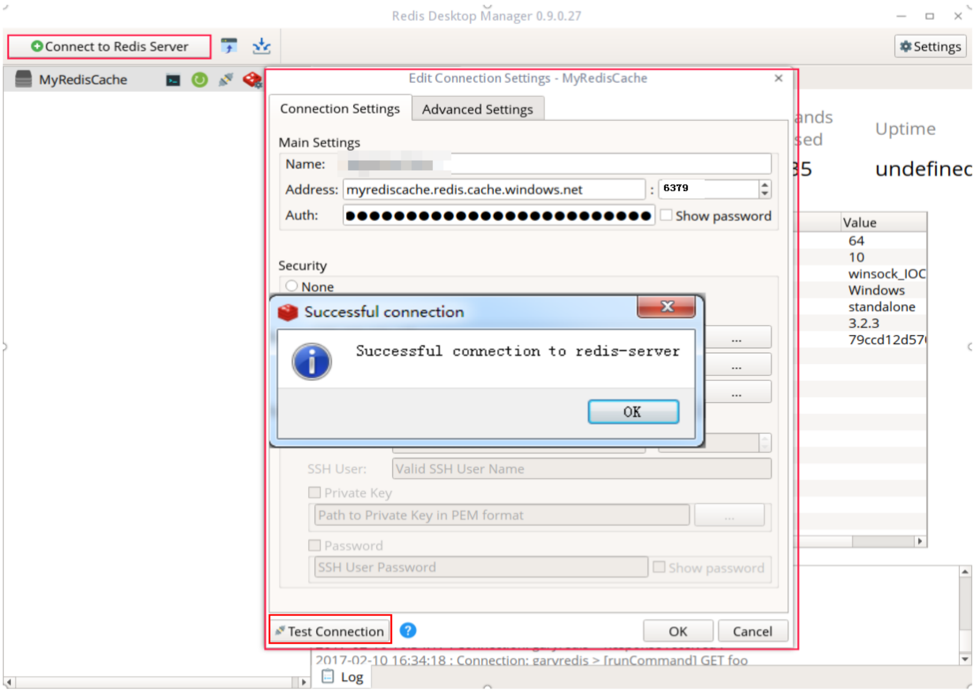
6.1.8 How Do I Access a DCS Redis Instance Through Redis Desktop Manager?

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 6-1 Accessing a DCS Redis instance through Redis Desktop Manager over the intranet



Redis Desktop Manager 0.9.0.27

Edit Connection Settings - MyRedisCache

Main Settings

Name: [redacted]

Address: myrediscache.redis.cache.windows.net : 6379

Auth: ●●●●●●●●●●●●●●●●●●●●●●●●● □ Show password

Security

○ None

Successful connection

Successful connection to redis-server

	Value
	64
	10
	winsock_IOC
	Windows
	standalone
	3.2.3
	79ccd12d57

2017-02-10 16:34:18 : Connection: garvredis > [runCommand] GET foo

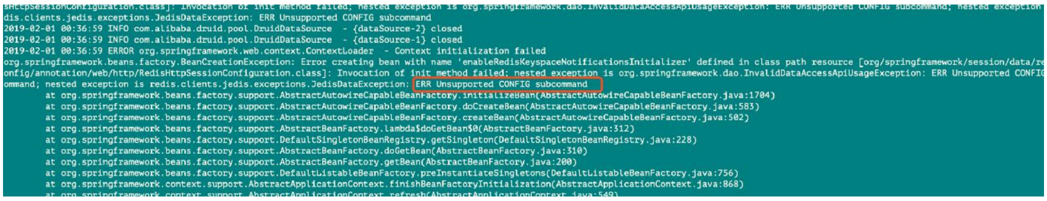
NOTE

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.

6.1.9 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 6-2 Spring Cloud error information



```
sHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG subcommand; nested exception
dis.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 INFO com.alibaba.druid.pool.DruidDataSource  - {dataSource-1} 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/re
onfig/annotation/web/http/RedisHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG
ommand; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1704)
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:583)
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:502)
        at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:312)
        at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:228)
        at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:310)
        at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:200)
        at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:756)
        at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:868)
        at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:549)
```

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 only by **single-node** and **master/standby** DCS Redis instances, but not by cluster DCS Redis instances.

6.1.10 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.

- 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**.

- 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.

6.1.11 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.

6.2 Redis Usage

6.2.1 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.

6.2.2 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**.

6.2.3 Why Aren't Security Groups Configured for DCS Redis 4.0 and 5.0 Instances?

Currently, DCS Redis 4.0 and Redis 5.0 instances use VPC endpoints and do not support security groups.

6.2.4 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.

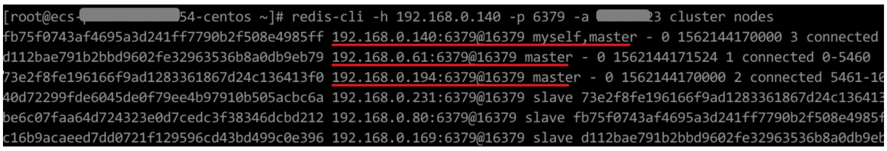
6.2.5 Can I Obtain the Addresses of the Nodes in a Cluster DCS Redis Instance?

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 5.0 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-[redacted]54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a [redacted]3 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 5461-10
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

6.2.6 Why Is Available Memory of a DCS Redis 3.0 Instance Smaller Than Instance Cache Size?

DCS Redis 3.0 instances are deployed on VMs and some memory is reserved for system overheads.

6.2.7 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).

Cluster DCS instances do not support data read/write in multiple databases.

6.2.8 Does DCS for Redis Support Redis Clusters?

Yes. DCS for Redis 4.0 and 5.0 support Redis Clusters. DCS for Redis 3.0 supports Proxy Clusters.

6.2.9 Does DCS for Redis Support Sentinel?

Yes. Redis Sentinel is supported by DCS for Redis 4.0 and 5.0 and is enabled by default. Sentinel constantly checks if master and replica nodes are running properly. If the master is not running properly, Sentinel starts a failover process and promotes a replica to master.

However, DCS for Redis 3.0 does not support Redis Sentinel. Instead, it uses keepalive to monitor master and replica nodes and to manage failovers.

6.2.10 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. In the current versions of DCS, you can select an eviction policy.

noeviction is the default eviction policy for single-node and master/standby DCS Redis instances. You can change the eviction policy by configuring the instance parameters on the DCS console.

volatile-lru is the default eviction policy for cluster DCS Redis instances. To change the eviction policy for cluster instances, contact the administrator.

When **maxmemory** is reached, you can select one of the following six 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.

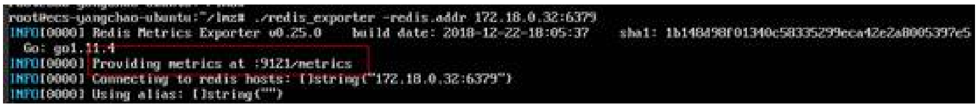
- **allkeys-lru**: DCS instances try to evict the least recently used keys first, in order to make space for new data.
- **volatile-lru**: DCS instances try to evict the least recently used keys with an expire set first, in order to make space for new data.
- **allkeys-random**: DCS instances recycle 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.

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**.

6.2.11 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-yangchao-ubuntu:~/lmz# ./redis_exporter -redis.addr 172.18.0.32:6379
INFO[0000] Redis Metrics Exporter v0.25.0    build date: 2018-12-22-18:05:37    sha1: 1b148498f01340c58335299eca42e2a8005397e5
  Go: go1.11.4
INFO[0000] Providing metrics at :9121/metrics
INFO[0000] Connecting to redis hosts: []string{"172.18.0.32:6379"}
INFO[0000] Using alias: []string{""}
```

6.2.12 Why Is Memory Usage More Than 100%?

This is normal due to Redis functions (such as master/replica replication and lazyfree). When the memory becomes full, scale up the instance or remove unnecessary data.

6.2.13 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**, **PEXIPRE**, 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.

6.2.14 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 but cannot change the port for accessing a DCS Redis 4.0 or 5.0 instance.

- Redis 3.0

  Use port 6379 for intra-VPC access.

- Redis 4.0 and Redis 5.0

  You can specify a port (ranging from 1 to 65535) or use the default port (6379) for accessing a DCS Redis 4.0 or 5.0 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**.

6.2.15 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.

For details about accessing DCS instances through clients, see **Accessing a DCS Instance**.

6.2.16 Does DCS Support Cross-AZ Deployment?

Master/Standby and cluster DCS Redis instances can be deployed across availability zones (AZs).

- If instances 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.

6.2.17 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.

6.2.18 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, see **Command Compatibility**.
2. DCS for Redis 3.0 does not support Redis 5.0 commands, such as Stream commands.

6.2.19 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**.

6.2.20 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 the large key. Then, run the **info** command to check the output buffer size.

6.2.21 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.

By default, data is not evicted from DCS instances. You can modify the instance configuration parameters to adjust the eviction policy so that the instance can evict key values.

6.3 Redis Commands

6.3.1 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 5.0

  Data of a DCS Redis 4.0 or 5.0 instance can be cleared all at once on the console or by the **FLUSHDB** or **FLUSHALL** command in redis-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.

6.3.2 How Do I Rename High-Risk Commands?

Currently, you can only rename the following critical commands: **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL** for DCS Redis 4.0 and 5.0 instances.

You can rename the commands when creating an instance on the console.

6.3.3 Does DCS for Redis Support Pipelining?

Cross-region image pull over public network is supported. For DCS Redis 4.0 and 5.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

6.3.4 Does DCS for Redis Support the INCR and EXPIRE Commands?

Cross-region image pull over public network is supported. Yes. For more information about Redis command compatibility, see **Command Compatibility**.

6.3.5 Why Do I Fail to Execute Some Redis Commands?

Possible causes include the following:

- The command is incorrect.
- 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 the administrator 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 the administrator for the instance to be upgraded.

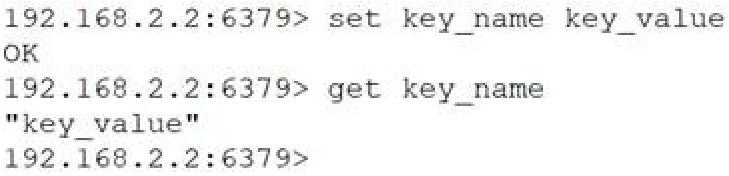
6.3.6 Why Does a Redis Command Fail to Take Effect?

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.

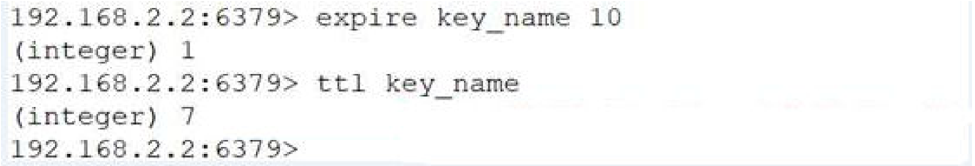
  The **SET** command is used to set the string value. If the value is not changed, run the following commands in redis-cli to access the instance:



```
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.

6.3.7 Is There a Time Limit on Executing Redis Commands? What Will Happen If a Command Times Out?

The time limit for executing a Redis command is 1 minute. This limit cannot be configured. After the execution of a command times out, your client will be automatically disconnected.

6.4 Instance Scaling and Upgrade

6.4.1 Can DCS Redis Instances Be Upgraded, for Example, from Redis 3.0 to Redis 4.0 or 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. 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**.

6.4.2 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.

6.4.3 Are Instance Resources Affected During Specification Modification?

No. Specification modifications can take place while the instance is running and do not affect any other resources.

6.4.4 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:

- Single-node and master/standby:

  The instance cannot be connected for several second and remains read-only for about 1 minute.

  For capacity expansion, only the memory of the instance is expanded. The CPU processing capability is not improved.

  Data of single-node instances may not be retained because they do not support data persistence. After the scaling, check whether the data is complete and import data if required.

- Proxy Cluster:

  The instance can be connected, but the CPU will be occupied and the latency will increase during data migration. During capacity expansion, new Redis Server nodes are added, and data is automatically balanced to the new nodes.

  Backup records created before the capacity change cannot be restored.

- Redis Cluster:

  The instance can be connected, but the CPU usage will increase and the latency will increase during data migration. During capacity expansion, new Redis Server nodes are added, and data is automatically balanced to the new nodes.

6.4.5 Why Do I Fail to Modify the Specifications for a DCS Redis Instance?

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 the administrator.

6.5 Monitoring and Alarm

6.5.1 Does Redis Support Command Audits?

No. To ensure high-performance reads and writes, Redis does not audit commands. Commands are not printed.

6.5.2 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 **http://www.redis.io/commands/info**.

6.5.3 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.

6.5.4 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.

6.6 Data Backup, Export, and Migration

6.6.1 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.

6.6.2 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 5.0

  Yes. Backup data of a DCS Redis 4.0 or 5.0 instance is exported from the console to RDB files. You cannot use redis-cli to export such data to RDB files.

6.6.3 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.

6.6.4 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.

6.6.5 Does DCS Support Data Persistence?

DCS Redis instances:

- Single-node: Not supported
- Master/Standby and cluster: Supported

6.7 Master/Standby Switchover

6.7.1 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.

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.

6.7.2 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.

6.7.3 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.

6.7.4 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.

A Change History

Table A-1 Change history

Released On	Description
2020-09-30	This issue is the first official release.