Distributed Cache Service

FAQs

Issue   01
Date   2024-06-19

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 Instance Types/Versions

1.1 New Features of DCS for Redis 4.0

Compared with DCS for Redis 3.0, DCS for Redis 4.0 and later versions add support for the new features of open-source Redis and supports faster instance creation.

Instance deployment changed from the VM mode to physical server–based containerization mode. An instance can be created within 8 to 10 seconds.

Redis 4.0 provides the following new features:

1. New commands, such as **MEMORY** and **SWAPDB**
2. Lazyfree, delaying the deletion of large keys and reducing the impact of the deletion on system resources
3. Memory performance optimization, that is, active defragmentation

MEMORY Command

In Redis 3.0 and earlier versions, you can execute the **INFO MEMORY** command to learn only the limited memory statistics. Redis 4.0 introduces the **MEMORY** command to help you better understand Redis memory usage.

```
127.0.0.1:6379[8]> memory help
1) MEMORY <subcommand> arg arg ... arg. Subcommands are:
2) DOCTOR - Return memory problems reports.
3) MALLOC-STATS -- Return internal statistics report from the memory allocator.
4) PURGE -- Attempt to purge dirty pages for reclamation by the allocator.
5) STATS -- Return information about the memory usage of the server.
6) USAGE <key> [SAMPLES <count>] -- Return memory in bytes used by <key> and its value. Nested values
are sampled up to <count
> times (default: 5).
127.0.0.1:6379[8]>
```

usage

Enter **memory usage** *[key]*. If the key exists, the estimated memory used by the value of the key is returned. If the key does not exist, **nil** is returned.

```
127.0.0.1:6379[8]> set dcs "DCS is an online, distributed, in-memory cache service compatible with Redis,
and Memcached."
OK
127.0.0.1:6379[8]> memory usage dcs
```

```
(integer) 141
127.0.0.1:6379[8]>
```

 📖 **NOTE**

1. **usage** collects statistics on the memory usage of the value and the key, excluding the Expire memory usage of the key.

   ```
   // The following is verified based on Redis 5.0.2. Results may differ in other Redis versions.
   192.168.0.66:6379> set a "Hello, world!"
   OK
   192.168.0.66:6379> memory usage a
   (integer) 58
   192.168.0.66:6379> set abc "Hello, world!"
   OK
   192.168.0.66:6379> memory usage abc
   (integer) 60   //After the key name length changes, the memory usage also changes. This indicates
   that the usage statistics contain the usage of the key.
   192.168.0.66:6379> expire abc 1000000
   (integer) 1
   192.168.0.66:6379> memory usage abc
   (integer) 60   // After the expiration time is added, the memory usage remains unchanged. This
   indicates that the usage statistics do not contain the expire memory usage.
   192.168.0.66:6379>
   ```

2. For hashes, lists, sets, and sorted sets, the **MEMORY USAGE** command samples statistics and provides the estimated memory usage.

   Usage: **memory usage** *keyset* **samples** *1000*

   keyset indicates the key of a set, and *1000* indicates the number of samples.

stats

Returns the detailed memory usage of the current instance.

Usage: **memory stats**

```
127.0.0.1:6379[8]> memory stats
 1) "peak.allocated"
 2) (integer) 2412408
 3) "total.allocated"
 4) (integer) 2084720
 5) "startup.allocated"
 6) (integer) 824928
 7) "replication.backlog"
... ...
```

The following table describes the meanings of some return items.

Table 1-1 MEMORY STATS return values

Return Value	Description
peak.allocated	Peak memory allocated by the allocator during Redis instance running. It is the same as **used_memory_peak** of **info memory**.
total.allocated	The number of bytes allocated by the allocator. It is the same as **used_memory** of **info memory**
startup.allocated	Initial amount of memory consumed by Redis at startup in bytes
replication.backlog	Size in bytes of the replication backlog. It is specified in the **repl-backlog-size** parameter. The default value is **1 MB**.

Return Value	Description
clients.slaves	The total size in bytes of all replicas overheads
clients.normal	The total size in bytes of all clients overheads
overhead.total	The sum of all overheads. **overhead.total** is the total memory **total.allocated** allocated by the allocator minus the actual memory used for storing data.
keys.count	The total number of keys stored across all databases in the server
keys.bytes-per-key	Average number of bytes occupied by each key. Note that the overhead is also allocated to each key. Therefore, this value does not indicate the average key length.
dataset.bytes	Memory bytes occupied by Redis data, that is, **overhead.total** subtracted from **total.allocated**
dataset.percentage	The percentage of **dataset.bytes** out of the net memory usage
peak.percentage	The percentage of peak.allocated out of **total.allocated**
fragmentation	Memory fragmentation rate

doctor

Usage: **memory doctor**

If the value of **used_memory** (**total.allocated**) is less than 5 MB, **MEMORY DOCTOR** considers that the memory usage is too small and does not perform further diagnosis. If any of the following conditions is met, Redis provides diagnosis results and suggestions:

1. The peak allocated memory is greater than 1.5 times of the current **total_allocated**, that is, **peak.allocated**/**total.allocated** > 1.5, indicating that the memory fragmentation rate is high, and that the RSS is much larger than **used_memory**.

2. The value of high fragmentation/fragmentation is greater than 1.4, indicating that the memory fragmentation rate is high.

3. The average memory usage of each normal client is greater than 200 KB, indicating that the pipeline may be improperly used or the Pub/Sub client does not process messages in time.

4. The average memory usage of each slave client is greater than 10 MB, indicating that the write traffic of the master is too high.

purge

Usage: **memory purge**

Executes the **jemalloc** internal command to release the memory. The released objects include the memory that is occupied but not used by Redis processes, that is, memory fragments.

◫ **NOTE**

> **MEMORY PURGE** applies only to the Redis instance that uses **jemalloc** as the allocator.

Lazyfree

Problem

Redis is single-thread. When a time-consuming request is executed, all requests are queued. Before the request is completed, Redis cannot respond to other requests. As a result, performance problems may occur. One of the time-consuming requests is deleting a large key.

Principle

The Lazyfree feature of Redis 4.0 avoids the blockage caused by deleting large keys, ensuring performance and availability.

When deleting a key, Redis asynchronously releases the memory occupied by the key. The key release operation is processed in the sub-thread of the background I/O (BIO).

Usage

1. Active deletion
   – **UNLINK**

     Similar to **DEL**, this command removes keys. If there are more than 64 elements to be deleted, the memory release operation is executed in an independent BIO thread. Therefore, the **UNLINK** command can delete a large key containing millions of elements in a short time.
   – **FLUSHALL** and **FLUSHDB**

     An **ASYNC** option was added to **FLUSHALL** and **FLUSHDB** in order to let the entire dataset or a single database to be freed asynchronously.
2. Passive deletion: deletion of expired keys and eviction of large keys

   There are four scenarios for passive deletion and each scenario corresponds to a parameter. These parameters are disabled by default.

   ```
   lazyfree-lazy-eviction no // Whether to enable Lazyfree when the Redis memory usage reaches
   maxmemory and the eviction policy is set.
   lazyfree-lazy-expire no // Whether to enable Lazyfree when the key with TTL is going to expire.
   lazyfree-lazy-server-del no // An implicit DEL key is used when an existing key is processed.
   slave-lazy-flush no // Perform full data synchronization for the standby node. Before loading the RDB
   file of the master, the standby node executes the FLUSHALL command to clear its own data.
   ```

Other New Commands

1. **swapdb**

   Swaps two Redis databases.

   swapdb *dbindex1 dbindex2*
2. **zlexcount**

   Returns the number of elements in the sorted set.

zlexcount *key min max*

Memory and Performance Optimization

1. Compared to before, the same amount of data can be stored with less memory.

2. Used memory can be defragmented and gradually evicted.

1.2 New Features of DCS for Redis 5.0

DCS for Redis 5.0 is compatible with the new features of the open-source Redis 5.0, in addition to all the improvements and new commands in Redis 4.0.

Stream Data Structure

Stream is a new data type introduced with Redis 5.0. It supports message persistence and multicast.

Figure 1-1 shows the structure of a Redis stream, which allows messages to be appended to the stream.

Streams have the following features:

1. A stream can have multiple consumer groups.

2. Each consumer group contains a **Last_delivered_id** that points to the last consumed item (message) in the consumer group.

3. Each consumer group contains multiple consumers. All consumers share the **last_delivered_id** of the consumer group. A message can be consumed by only one consumer.

4. **pending_ids** in the consumer can be used to record the IDs of items that have been sent to the client, but have not been acknowledged.

5. For detailed comparison between stream and other Redis data structures, see **Table 1-2**.

Figure 1-1 Stream data structure
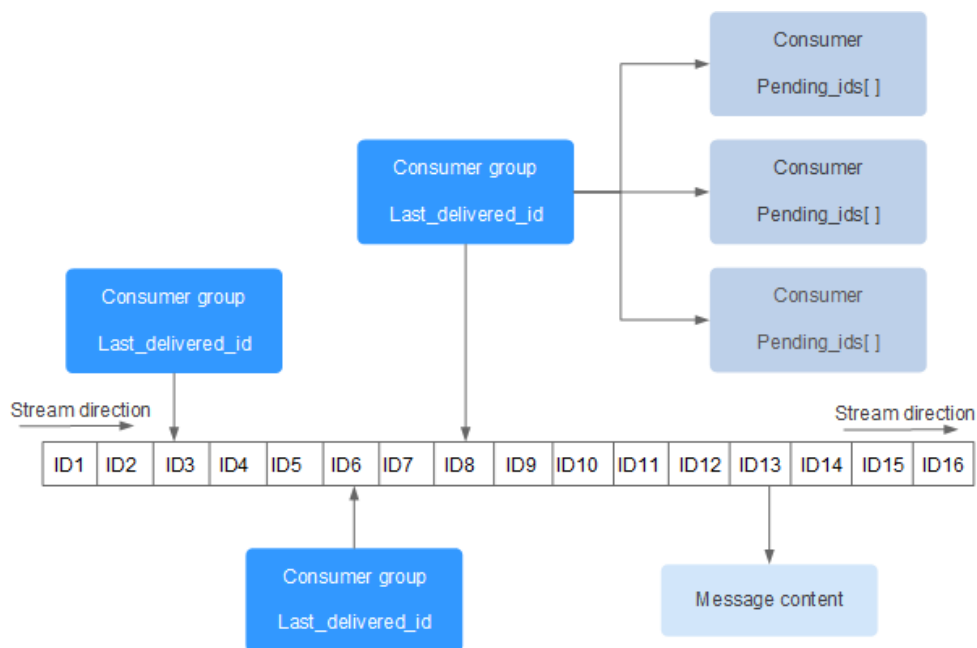
Table 1-2 Differences between streams and existing Redis data structures

Item	Stream	List, Pub/Sub, Zset
Complexity of seeking items	O(log(N))	List: O(N)
Offset	Supported. Each item has a unique ID. The ID is not changed as other items are added or evicted.	List: Not supported. If an item is evicted, the latest item cannot be located.
Persistence	Supported. Streams are persisted to AOF and RDB files.	Pub/Sub: Not supported.
Consumer group	Supported.	Pub/Sub: Not supported.
Acknowledgment	Supported.	Pub/Sub: Not supported.
Performance	Not related to the number of consumers.	Pub/Sub: Positively related to the number of clients.
Eviction	Streams are memory efficient by blocking to evict the data that is too old and using a radix tree and listpack.	Zset consumes more memory because it does not support inserting same items, blocking, or evicting data

Item	Stream	List, Pub/Sub, Zset
Randomly deleting items	Not supported.	Zset: Supported.

Stream commands

Stream commands are described below in the order they are used. For details, see **Table 1-3**.

1. Run the **XADD** command to add a stream item, that is, create a stream. The maximum number of messages that can be saved can be specified when adding the item.
2. Create a consumer group by running the **XGROUP** command.
3. A consumer uses the **XREADGROUP** command to consume messages.
4. After the consumption, the client runs the **XACK** command to confirm that the consumption is successful.

Figure 1-2 Stream commands
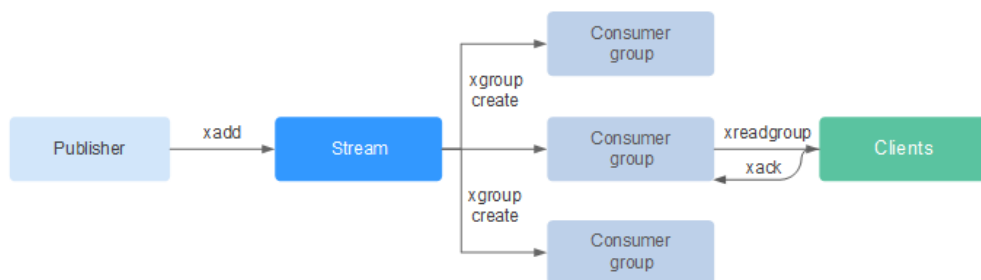
Table 1-3 Stream commands description

Command	Description	Syntax
XACK	Deletes one or multiple messages from the *pending entry list* (PEL) a consumer group of the stream.	XACK key group ID [ID ...]
XADD	Adds a specified entry to the stream at a specified key. If the key does not exist, running this command will result in a key to be automatically created based on the entry.	XADD key ID field string [field string ...]
XCLAIM	Changes the ownership of a pending message, so that the new owner is the consumer specified as the command argument.	XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]

Command	Description	Syntax
XDEL	Removes the specified entries from a stream, and returns the number of entries deleted, that may be different from the number of IDs passed to the command in case certain IDs do not exist.	XDEL key ID [ID …]
XGROUP	Manages the consumer groups associated with a stream You can use **XGROUP** to: ● Create a new consumer group associated with a stream. ● Destroy a consumer group. ● Remove a specified consumer from a consumer group. ● Set the consumer group *last delivery ID* to something else.	XGROUP [CREATE key groupname id-or-$] [SETID key id-or-$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]
XINFO	Retrieves different information about the streams and associated consumer groups.	XINFO [CONSUMERS key groupname] key key [HELP]
XLEN	Returns the number of entries in a stream. If the specified key does not exist, **0** is returned, indicating an empty stream.	XLEN key
XPENDING	Obtains data from a stream through a consumer group. This command is the interface to inspect the list of pending messages in order to observe and understand what clients are active, what messages are pending to be consumed, or to see if there are idle messages.	XPENDING key group [start end count] [consumer]
XRANGE	Returns entries matching a given range of IDs.	XRANGE key start end [COUNT count]
XREAD	Reads data from one or multiple streams, only returning entries with an ID greater than the last received ID reported by the caller.	XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key …] ID [ID …]
XREADGROUP	A special version of the **XREAD** command, which is used to specify a consumer group to read from.	XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] STREAMS key [key …] ID [ID …]

Command	Description	Syntax
XREVRANGE	This command is exactly like **XRANGE**, but with the notable difference of returning the entries in reverse order, and also taking the start-end range in reverse order.	XREVRANGE key end start [COUNT count]
XTRIM	Trims the stream to a specified number of items, if necessary, evicting old items (items with lower IDs).	XTRIM key MAXLEN [~] count

Message (stream item) acknowledgement

Compared with Pub/Sub, streams not only support consumer groups, but also message acknowledgement.

When a consumer invokes the **XREADGROUP** command to read or invokes the **XCLAIM** command to take over a message, the server does not know whether the message is processed at least once. Therefore, once having successfully processed a message, the consumer should invoke the **XACK** command to notify the stream so that the message will not be processed again. In addition, the message is removed from PEL and the memory will be released from the Redis server.

In some cases, such as network faults, the client does not invoke **XACK** after consumption. In such cases, the item ID is retained in PEL. After the client is reconnected, set the start message ID of **XREADGROUP** to 0-0, indicating that all PEL messages and messages after **last_id** are read. In addition, repeated message transmission must be supported when consumers consume messages.

Figure 1-3 Acknowledgment mechanism
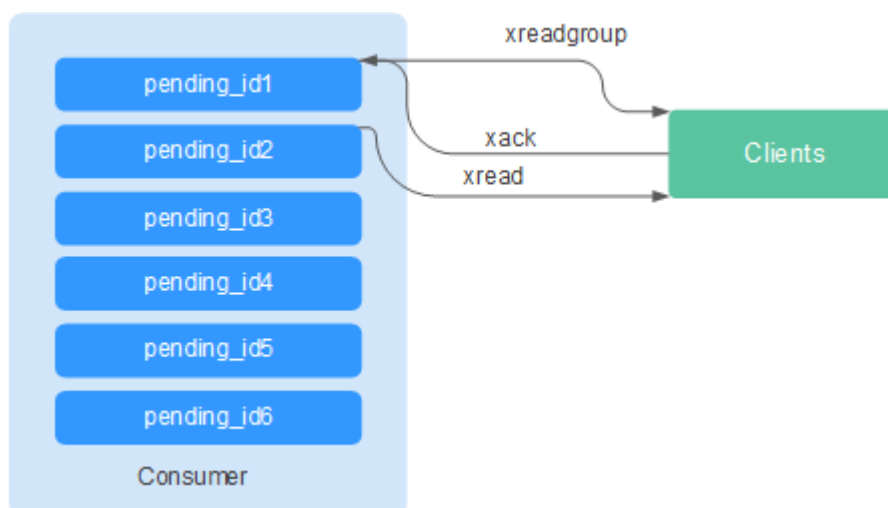
Memory Usage Optimization

The memory usage of Redis 5.0 is optimized based on the previous version.

- Active defragmentation

  If a key is modified frequently and the value length changes constantly, Redis will allocate additional memory for the key. To achieve high performance, Redis uses the memory allocator to manage memory. Memory is not always freed up to the OS. As a result, memory fragments occur. If the fragmentation ratio (**used_memory_rss**/**used_memory**) is greater than 1.5, the memory usage is inefficient.

  To reduce memory fragments, properly plan and use cache data and standardize data writing.

  For Redis 3.0 and earlier versions, memory fragmentation problems are resolved by restarting the process regularly. It is recommended that the actual cache data does not exceed 50% of the available memory.

  For Redis 4.0, active defragmentation is supported, and memory is defragmented while online. In addition, Redis 4.0 supports manual memory defragmentation by running the **memory purge** command.

  For Redis 5.0, improved active defragmentation is supported with the updated Jemalloc, which is faster, more intelligent, and provides lower latency.

- HyperLogLog implementation improvements

  A HyperLogLog is a probabilistic data structure used to calculate the cardinality of a set while consuming little memory. Redis 5.0 improves HyperLogLog by further optimizing its memory usage.

  For example: the B-tree is efficient in counting, but consumes a lot of memory. By using HyperLogLog, a lot of memory can be saved. While the B-tree requires 1 MB memory for counting, HyperLogLog needs only 1 KB.

- Enhanced memory statistics

  The information returned by the **INFO** command is more detailed.

New and Better Commands

1. **Enhanced client management**

   - redis-cli supports cluster management.

     In Redis 4.0 and earlier versions, the **redis-trib** module needs to be installed to manage clusters.

     Redis 5.0 optimizes redis-cli, integrating all cluster management functions. You can run the **redis-cli --cluster help** command for more information.

   - The client performance is enhanced in frequent connection and disconnection scenarios.

     This optimization is valuable when your application needs to use short connections.

2. **Simpler use of sorted sets**

   ZPOPMIN and **ZPOPMAX** commands are added for sorted sets.

   - ZPOPMIN key [count]

     Removes and returns up to **count** members with the lowest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with higher scores.

– ZPOPMAX key [count]

Removes and returns up to **count** members with the highest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with lower scores.

3. **More sub-commands added to the help command**

The **help** command can be used to view help information, saving you the trouble of visiting **redis.io** every time. For example, run the following command to view the stream help information: **xinfo help**

```
127.0.0.1:6379> xinfo help
1) XINFO <subcommand> arg arg ... arg. Subcommands are:
2) CONSUMERS <key> <groupname>  -- Show consumer groups of group <groupname>.
3) GROUPS <key>           -- Show the stream consumer groups.
4) STREAM <key>           -- Show information about the stream.
5) HELP              -- Print this help.
127.0.0.1:6379>
```

4. **redis-cli command input tips**

After you enter a complete command, redis-cli displays a parameter tip to help you memorize the syntax format of the command.

As shown in the following figure, run the **zadd** command, and redis-cli displays **zadd** syntax in light color.

```
# Cluster
cluster_enabled:0

# Keyspace
db0:keys=1,expires=0,avg_ttl=0
198.19.59.199:6379> zadd key [NX|XX] [CH] [INCR] score member [score member ...]
```

RDB Storing LFU and LRU Information

In Redis 5.0, storage key eviction policies **LRU** and **LFU** were added to the RDB snapshot file.

● FIFO: First in, first out. The earliest stored data is evicted first.

● LRU: Least recently used. Data that is not used for a long time is evicted first.

● LFU: Least frequently used. Data that is least frequently used is evicted first.

◫ NOTE

The RDB file format of Redis 5.0 is modified and is backward compatible. Therefore, if a snapshot is used for migration, data can be migrated from the earlier Redis versions to Redis 5.0, but cannot be migrated from the Redis 5.0 to the earlier versions.

1.3 New Features of DCS for Redis 6.0

DCS for Redis 6.0 is compatible with the new features of the open-source Redis 6.0, in addition to all the improvements and commands in Redis 5.0.

RESP3

Redis 6.0 introduced RESP3, the next-generation Redis protocol which implements more data types than RESP2.

- **Null**: a non-existent value. It replaces RESP2's ***-1** and **$-1**.
- **Array**: an ordered collection
- **Simple string**: a space-efficient non-binary safe string
- **Blob string**: a binary safe string
- **Simple error**: a space-efficient non-binary safe error code or message
- **Blob error**: binary safe error code or message
- **Boolean**: true or false
- **Number**: a 64-bit signed integer
- **Big number**: a large number
- **Double**: a floating point number
- **Verbatim string**: a binary safe string with a text format
- **Map**: an unordered collection of key-value pairs
- **Set**: an unordered collection of non-repeated elements
- **Attribute**: attribute key-value pairs, similar to Map
- **Push**: out-of-band data, similar to Array, used by the Redis server to push data to the client.
- **Hello**: response returned by the **HELLO** command, sent when the connection between the client and the server is established.

📖 **NOTE**

To use RESP3, ensure that the client SDK supports it. Otherwise, the client SDK communicates with the server through **HELLO** of RESP2.

Client-side Caching

Redis 6.0 uses tracking to proactively instruct clients to refresh their cache. There are three implementation modes:

RESP3

- Default mode
- Broadcasting mode

RESP2

- Redirecting mode

Format for enabling client-side cache tracking:

```
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix] [BCAST] [OPTIN][OPTOUT] [NOLOOP]
```

In RESP3, the server uses Push messages to send notifications. In the default mode, Redis remembers the keys requested by each client. When the value of a key changes, Redis sends an invalidation message to notify the corresponding client. Note that Redis notifies each client only once. If the value changes again later, the client must read the key again to enable tracking. To enable tracking in the default mode, use the following command:

```
CLIENT TRACKING ON
```

In the broadcasting mode, all clients are notified when the value of the key matching a tracked key prefix changes. If there is large number of keys that match

the key prefix or there is large number of changes, the server sends many invalidation messages, consuming much network bandwidth. To enable tracking in the broadcasting mode, use the following command:

```
CLIENT TRACKING ON BCAST PREFIX key-prefix
```

If the client SDK does not support RESP3, only the redirecting mode of RESP2 can be used to implement tracking. In this case, a dedicated client that supports RESP3 needs to act as the transit node that redirects invalidation messages from Redis to specific subscription channels.

Figure 1-4 Client cache tracking (redirecting mode)
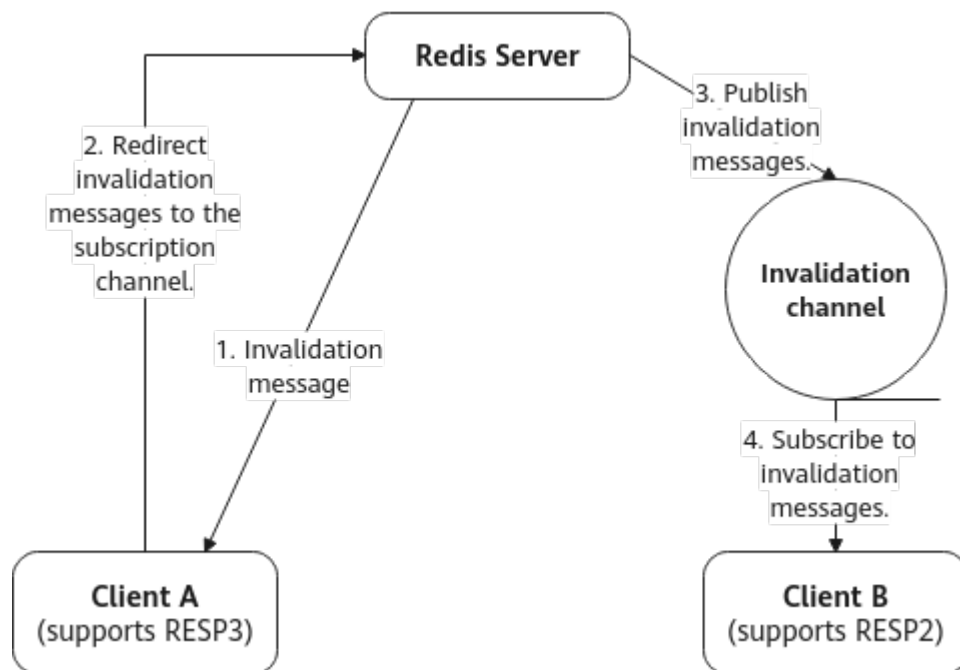
Faster RDB Loading

In Redis 6.0, the loading of RDB files is optimized for a 20% to 30% increase in speed.

Optimized INFO Command

The processing of the **INFO** command is optimized, especially in the scenario where a large number of clients are connected, resulting in higher performance and lower latency.

1.4 How Do I View the Version of a DCS Redis Instance?

Connect to the instance and run the **INFO** command.

Figure 1-5 Querying instance information


```
> INFO
  # Server

  redis_version:5.0.14

  patch_version:5.0.14.1

  redis_git_sha1:00000000

  redis_git_dirty:0
```

2 Feature

2.1 What Are the CPU Specifications of DCS Instances?

When using DCS, you only need to pay attention to critical indicators such as QPS, bandwidth, and memory. You do not need to be concerned about CPU specifications.

DCS for Redis is based on open-source Redis. Open-source Redis uses a single main thread to process commands, so only one CPU core is used on each Redis node. Increasing the memory of DCS Redis instances does not change CPU specifications.

Due to this restriction, **you can use a cluster instance and add shards to achieve higher CPU performance**. Each node in a cluster uses one vCPU by default.

2.2 What Are Shard and Replica Quantities?

Shard

A **shard** is a management unit in Redis clusters. 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. Shards increase cache capacity and concurrent connections.

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

Replica

A replica refers to a **node** of a DCS instance. It can be a master node or a standby node. A single-replica instance has no standby node. A two-replica instance has one master node and one standby node. For example, if the number of replicas is set to three for a master/standby instance, the instance has one master node and two standby nodes.

Number of Replicas and Shards of Different Instance Types

- **Single-node**: Each instance has only one node (one Redis process). If the Redis process is faulty, DCS starts a new Redis process for the instance.
- **Master/standby and read/write splitting**: Each instance has one shard, which contains one master node and one or more standby nodes. If the master node is faulty, master/standby switchover is triggered to restore services. The more the replicas (standby nodes), the better the reliability (performance is unaffected).
- **Cluster**: Each instance has multiple shards. By default, each shard is a master/standby instance with two replicas. For example, if a cluster instance has three shards and three replicas, each shard has three nodes (one master node and two standby nodes).

Instance Type	Shards	Replicas	Load Balancing	IP Addresses
Single-node	1	1 (only)	-	1
Master/ Standby	1	2 (default) DCS Redis 4.0/5.0 Master/ Standby instances support 2 to 10 replicas. Master/ Standby instances of other versions support only 2 replicas.	Not supported	Same as the number of replicas
Read/write splitting	1	Default: 2; max.: 6	Supported	1
Proxy Cluster	Multiple	2 (not customizable)	Supported	1
Redis Cluster	Multiple	2 (default) DCS Redis 4.0/5.0 Redis Cluster instances support 1 to 5 replicas. Redis Cluster 6.0 instances support 1 to 2 replicas.	Not supported	Number of replicas x Number of shards

2.3 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.

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

For a cluster DCS Redis 4.0 or later instance (Redis Cluster type), run the **CLUSTER NODES** command to obtain node addresses:

redis-cli -h *{redis_address}* **-p** *{redis_port}* **-a** *{redis_password}* **cluster nodes**

In the output similar to the following, obtain the IP addresses and port numbers of all the master nodes.

2.5 Does DCS for Redis Support Redis Clusters?

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

2.6 What Are Redis Data Eviction Policies?

Data is evicted from cache based on a user-defined space limit in order to make space for new data. For details, see the **Redis official website**. You can **view or change the eviction policy** by configuring an instance parameter on the DCS console.

Eviction Policies Supported by DCS Redis Instances

When **maxmemory** is reached, you can select one of the following eight eviction policies:

- **noeviction**: When the memory limit is reached, DCS instances return errors to clients and no longer process write requests and other requests that could result in more memory to be used. However, **DEL** and a few more exception requests can continue to be processed.

- **allkeys-lru**: DCS instances try to evict the least recently used keys first, in order to make space for new data.

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

- **allkeys-random**: DCS instances recycle random keys so that new data can be stored.

- **volatile-random**: DCS instances evict random keys with an expire set, in order to make space for new data.

- **volatile-ttl**: DCS instances evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first, in order to make space for new data.

- **allkeys-lfu**: DCS instances evict the least frequently used keys from all keys.

- **volatile-lfu**: DCS instances evict the least frequently used keys with an **expire** field from all keys.

📖 **NOTE**

- If no key can be recycled, **volatile-lru**, **volatile-random**, and **volatile-ttl** are the same as **noeviction**. For details, see the description of **noeviction**.
- The default eviction policy is **volatile-lru**.

Viewing or Changing Eviction Policies

You can view or change the eviction policy with the **maxmemory-policy** parameter.
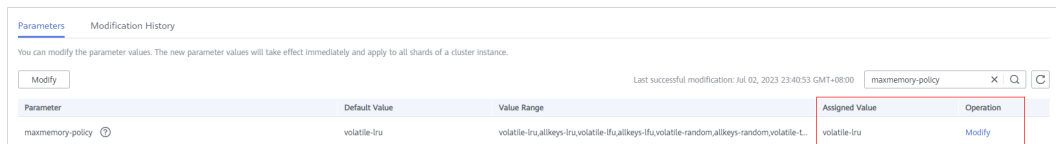
2.7 Does DCS for Redis Support Read/Write Splitting?

The following table describes DCS's support for read/write splitting.

Instance Type	Read/Write Splitting
Read/write splitting	Supported. **NOTE** To implement read/write splitting without client configurations, use **read/write splitting instances**.
Redis Cluster	Read/write splitting can be configured and implemented on the client. For details, see **Configuration**.
Master/Standby Redis	Read/write splitting can be implemented on a client that is able to distinguish between read and write requests.

Instance Type	Read/Write Splitting
Other	Not supported.

Configuration

- For a **Redis Cluster instance**, you can query all master and replica nodes by running the **CLUSTER NODES** command. The client will connect to replicas and configure read-only access on them.

    Run the following command to query cluster nodes:

    `redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes`

    Read-only configuration on replicas is achieved through the **READONLY** command.

- **For a master/standby instance**, there are two domain names displayed on the instance details page of the console: a read/write address (master node) and a read-only address (standby node). On the client, you can direct write requests to the read/write domain name and read requests to the read-only domain name.

- For a **read/write splitting instance**, read/write splitting is implemented on the server side by default. Proxies distinguish between read and write requests, and forward write requests to the master node and read requests to the standby node. You do not need to perform any configuration on the client.

2.8 Does DCS for Redis Support Multi-DB?

DCS's support for multiple databases (multi-DB) is as follows:

- Single-node, read/write splitting, and master/standby DCS Redis instances: Multi-DB is supported. By default, there are 256 databases, numbering 0–255. The default database is DB0. Multi-DB is used for data isolation. The size of each database is not evenly allocated. As a result, one database may fully occupy the memory of the instance.

- Proxy Cluster: There is only one database by default.
    - For details about how to create a Proxy Cluster instance with multiple databases, see **How Do I Create a Multi-DB Proxy Cluster Instance?**
    - For details about how to enable multi-DB for a single-DB Proxy Cluster instance, see **What Are the Constraints on Implementing Multiple Databases on a Proxy Cluster Instance?**

- Redis Cluster DCS instances: Multi-DB is not supported. There is only one database (DB0).

The number of databases cannot be changed, and the size of each database cannot be customized.

2.9 Does DCS Support External Extensions, Plug-ins, or Modules?

No. DCS for Redis does not support external extensions, plug-ins, or modules. There is no plan for supporting modules.

2.10 Does DCS Support Data Persistence? What Is the Impact of Persistence?

Support for Persistence

- DCS Redis instances:
  - Single-node: Data persistence is not supported.
  - Master/standby, read/write splitting, and cluster (except single-replica clusters): Data persistence is supported.
- DCS Memcached instances:
  - Single-node: Data persistence is not supported.
  - Master/standby: Data persistence is supported.

Persistence Modes

- DCS supports only AOF persistence by default. You can enable or disable persistence as required. All instances except single-node and single-replica cluster ones are created with AOF persistence enabled.
- DCS does not support RDB persistence by default and you cannot configure the **save** parameter. If RDB persistence is required for a master/standby or cluster instance of Redis 4.0 or later, you can use the backup and restoration function to back up the instance data to an RDB file and store the data in OBS.

Disk Used for Persistence

For DCS Redis 4.0 and later instances, data is persisted to SSD disks.

Impact of AOF Persistence

After AOF persistence is enabled, the Redis-Server process needs to record operations in the AOF file for data persistence.

- If the disk or I/O of the underlying compute node is faulty, the latency may increase or a master/standby switchover may occur.
- Redis-Server periodically rewrites the AOF. During a rewrite, the latency may be high for a short time. For details about the AOF rewriting rules, see **When Will AOF Rewrites Be Triggered?**

If DCS instances are used for application acceleration, you are advised to disable AOF persistence for higher performance and stability.

Exercise caution when disabling AOF persistence. After it is disabled, cached data may be lost in extreme scenarios, for example, when both the master and standby nodes are faulty.

To disable AOF persistence, set parameter **appendonly** to **no** on the instance details page.

2.11 Do DCS Redis Instances Limite Data Quantity?

No, but the data volume cannot exceed the memory of a DCS instance.

3 Security

3.1 How Do I Configure a Security Group?

Security Group of a Migration Task

- When creating an online migration task, select a security group. Its outbound rules must allow traffic over the IP addresses and ports of the source and target Redis instances. By default, all outbound traffic is allowed.
- Backup import uses the **default** security group. Ensure that all outbound traffic is allowed (this is the default setting.)

Figure 3-1 Outbound rules of the migration security group

3.2 Why Can't Security Groups Be Configured for DCS Redis 4.0/5.0/6.0 Basic Edition Instances?

Currently, DCS Redis 4.0/5.0/6.0 basic edition instances use VPC endpoints and do not support security groups.

To allow access only from specific IP addresses to a DCS Redis 4.0, 5.0, or 6.0 basic edition instance, add the IP addresses to the instance whitelist.

If no whitelists are added for the instance or the whitelist function is disabled, all IP addresses that can communicate with the VPC can access the instance.

Creating a Whitelist Group

Step 1 Log in to the management console, and choose **Application** > **Distributed Cache Service** in the service list.

Step 2 Click in the upper left corner of the management console and select the region where your instance is located.

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

Step 4 Click the name of a DCS instance.

Step 5 Choose **Instance Configuration** > **Whitelist**. On the displayed page, click **Create Whitelist Group**.

Step 6 In the **Create Whitelist Group** dialogue box, specify **Group Name** and **IP Address/Range**.

Table 3-1 Whitelist parameters

Parameter	Description	Example
Group Name	Whitelist group name of the instance. A maximum of four whitelist groups can be created for each instance.	DCS-test
IP Address/ Range	A maximum of 20 IP addresses or IP address ranges can be added to an instance. Separate multiple IP addresses or IP address ranges with commas. Unsupported IP address and IP address range: 0.0.0.0 and 0.0.0.0/0.	10.10.10.1,10.10.10.10

Step 7 Click **OK**.

The whitelist function takes effect immediately after the whitelist group is created. Only whitelisted IP addresses can access the instance. For persistent connections, the whitelist takes effect after reconnection.

◫ **NOTE**

- In the whitelist group list, click **Edit** to modify the IP addresses or IP address ranges in a group, and click **Delete** to delete a whitelist group.
- After whitelist has been enabled, you can click **Disable Whitelist** above the whitelist group list to allow all IP addresses connected to the VPC to access the instance.

----End

3.3 How Can I Secure My DCS Redis Instances?

Redis is one of the most powerful and widely used open-source cache technologies. However, the open-source Redis does not have robust security features of its own. It is vulnerable to malicious Internet attacks, possibly causing data breaches.

To secure your DCS Redis instances, consider taking the following advice:

- Network connection configurations

  a. Encrypt sensitive data.

     Sensitive data must be encrypted before being stored.

  b. Configure ECS firewalls.

     Configure firewall filtering rules for the ECS where your client runs.

  c. Set the instance password.

  d. Configure a whitelist.

- redis-cli usage

  a. Hide the password.

     Problem: If the **-a <password>** option is used, the password may show up when the **ps** command is run.

     Solution: Modify the Redis source code. Hide the password immediately after starting redis-cli by calling the main function.

  b. Disable sudo in running scripts.

     Problem: Parameters for starting redis-cli contain sensitive patterns related to the password, which may show up when the **ps** command is run and may be logged.

     Solution: Access the instance by calling APIs (or through redis-py in Python). Do not allow switching to the **dbuser** user using sudo in redis-cli.

3.4 Does DCS Support Cross-AZ Deployment?

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

- If instance nodes in an AZ are faulty, nodes in other AZs will not be affected. The standby node automatically becomes the master node to continue to operate, ensuring disaster recovery (DR).

- Cross-AZ deployment does not compromise the speed of data synchronization between the master and standby nodes.

3.5 Is a Password Required for Accessing an Instance? How Do I Set a Password?

- A DCS Redis instance can be access with or without a password. You can directly access a DCS Redis instance through a Redis client without setting a

password. However, for security purposes, you are advised to set a password for authentication and verification whenever possible. The password must be set when you create the instance.

- To change the Redis instance access mode, or change or reset a password, see **Managing Passwords**.

3.6 What Is Sentinel?

Overview

High availability in Redis is implemented through Sentinel. Sentinel helps you defend against certain types of faults without manual intervention, and complete tasks such as monitoring, notification, and client configuration. For details, see the **Redis official website**.

Principles

Redis Sentinel is a distributed system where multiple Sentinel processes work together. It has the following advantages:

1. Fault detection is performed only when multiple Sentinels agree that a master node is unavailable, which reduces the possibility of false positives.

2. Even if some Sentinel processes are faulty, the Sentinel system can still work properly to prevent faults.

On a higher level, there is a larger distributed system consisting of Sentinels, Redis master and replica nodes, and clients connected to Sentinels and Redis.

Functions

- Monitoring: Sentinel continuously checks whether the master and replica nodes are working properly.

- Notification: If a node is faulty, Sentinel can notify the system administrator or other computer programs by calling an API.

- Automatic failover: If the master node is abnormal, Sentinel starts a failover to promote a replica to master. Other replicas replicate data from the new master node. Applications that use the Redis instance will be notified that they should connect to the new address.

- Client configuration: Sentinel serves as the authoritative source for client service discovery. Clients connect to Sentinel and requests the address of the master Redis node that is responsible for specific services. If a failover occurs, Sentinels delivers the new address.

3.7 Does DCS Manage Instances in Sentinel Mode?

- Cluster instances, master/standby and read/write splitting DCS Redis 4.0, 5.0, and 6.0 instances support Sentinels. Sentinels monitor the running status of both the master and standby nodes of a master/standby instance and each shard of a cluster instance. If the master node becomes faulty, a failover will be performed and the service will be resumed automatically.

Sentinels are invisible to you and is used only in the service.

4 Client and Network Connection

4.1 Troubleshooting Redis Connection Failures

Overview

This topic describes why Redis connection problems occur and how to solve the problems.

Problem Classification

To troubleshoot abnormal connections to a Redis instance, check the following items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Connection Between the Redis Instance and the ECS

The ECS where the client is located must be in the same VPC as the Redis instance and be able to communicate with the Redis instance.

- For a DCS Redis 4.0/5.0/6.0 basic instance, check the whitelist of the instance.

  If the instance has a whitelist, **ensure that the client IP address is included in the whitelist**. Otherwise, the connection will fail. For details, see **Managing IP Address Whitelist**. If the client IP address changes, add the new IP address to the whitelist.

- Check the regions of the Redis instance and the ECS.

  If the Redis instance and the ECS are not in the same region, create another Redis instance in the same region as the ECS and migrate data from the old instance to the new instance by referring to **Data Migration Guide**.

- Check the VPCs of the Redis instance and the ECS.

  Different VPCs cannot communicate with each other. An ECS cannot access a Redis instance if they are in different VPCs. You can establish VPC peering connections to allow the ECS to access the Redis instance across VPCs.

  For more information on how to create and use VPC peering connections, see **VPC Peering Connection**.

Password

If the instance password is incorrect, the port can still be accessed but the authentication will fail. If you forget the password, you can reset the password. For details, see **Resetting Instance Passwords**.

Instance Configuration

If a connection to Redis is rejected, log in to the DCS console, go to the instance details page, and modify the **maxclients** parameter. For details, see **Modifying Configuration Parameters of an Instance**.

Client Connections

- The connection fails when you use redis-cli to connect to a Redis Cluster instance.

  Solution: Check whether **-c** is added to the connection command. Ensure that the correct connection command is used when connecting to the cluster nodes.

  – 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 **Access Using redis-cli**.

- Error "Read timed out" or "Could not get a resource from the pool" occurs.

  Solution:

  – 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 do not execute the command frequently.

- Error "unexpected end of stream" occurs and causes service exceptions.

  Solution:

  – Optimize the Jedis connection pool by referring to **Recommended Jedis Parameter Settings**.

  – Check whether there are many big keys. For details, see

- The connection is interrupted.

  Solution:

  – Modify the application timeout duration.

  – Optimize the service to avoid slow queries.

  – Replace the **KEYS** command with the **SCAN** command.

- If an error occurs when you use the Jedis connection pool, see **Troubleshooting a Jedis Connection Pool Error**.

Bandwidth

If the bandwidth reaches the upper limit of the corresponding instance specifications, Redis connections may time out.

You can view the **Flow Control Times** metric to check whether the bandwidth has reached the upper limit.

Then, check whether the instance has big keys and hot keys. If a single key is too large or overloaded, operations on the key may occupy too many bandwidth resources. For details about big keys and hot keys, see **Analyzing Big Keys and Hot Keys**.

Redis Performance

Connections to an instance may become slow or time out if the CPU usage spikes due to resource-consuming commands such as **KEYS**, or too much memory is used because the expiration time is not set for the instance or expired keys remain in the memory. In these cases, do as follows:

- Use the **SCAN** command instead of the **KEYS** command, or disable the **KEYS** command.

- Check the monitoring data and configure alarm rules. For details, see **Setting Alarm Rules for Critical Metrics**.

  For example, you can view the **Memory Usage** and **Used Memory** metrics to keep track of the instance memory usage, and view the **Connected Clients** metric to determine whether the instance connections limit has been reached.

- Check whether the instance has big keys and hot keys.

  For details about the operations of big key and hot key analysis, see **Analyzing Big Keys and Hot Keys**.

4.2 Does DCS Support Cross-VPC Access?

Cross-VPC means the client and the instance are not in the same VPC.

The following assumes that public access is disabled for a DCS instance. Generally, VPCs are isolated from each other and an ECS cannot access a DCS instance that belongs to a different VPC from the ECS.

You can establish VPC peering connections to allow the ECS to access the DCS instance across VPCs.

When using VPC peering connections to access DCS instances across VPCs, adhere to the following rules:

- If CIDR Blocks 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 CIDR Blocks: 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24.

- If CIDR Blocks 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 CIDR Blocks: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.

- If CIDR Blocks 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 CIDR Blocks: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.

For more information on how to create and use VPC peering connections, see **VPC Peering Connection**.

4.3 Why Is "(error) NOAUTH Authentication required" Displayed When I Access a DCS Redis Instance?

This is because you have enabled password-free access for the instance. To prevent the error message from appearing, do not enter any password.

4.4 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.

4.5 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.

When a connection times out, check whether AOF persistence is enabled for Redis. Then, determine whether to enable the function (learn about the **impact of AOF persistence**) as required. Disabling AOF persistence improves stability and reduces blocking on the client to prevent connection failures.

If timeout errors occur frequently, contact technical support.

4.6 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 JedisPool 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 Check the 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. If public access is enabled for your instance, check whether the IP address configured on the Jedis client is the same as the EIP bound to your instance. If they are inconsistent, modify the IP address configuration and then try again.

2. Test the network.

   Use the ping command and telnet on the client to test the network.

Step 2 Check the number of connections.

Check whether the number of established network connections exceeds the upper limit configured for JedisPool. 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.

If there are too many **TIME_WAIT** connections, modify the kernel parameters by running the **/etc/sysctl.conf** command as follows:

```
##Uses cookies to prevent some SYN flood attacks when the SYN waiting queue overflows.
net.ipv4.tcp_syncookies = 1
##Reuses TIME_WAIT sockets for new TCP connections.
net.ipv4.tcp_tw_reuse = 1
##Enables quick reclamation of TIME_WAIT sockets in TCP connections.
```

```
net.ipv4.tcp_tw_recycle = 1
##Modifies the default timeout time of the system.
net.ipv4.tcp_fin_timeout = 30
```

After the modification, run the **/sbin/sysctl -p** command for the modification to take effect.

Step 5 If the problem persists after you perform the preceding checks, perform the following steps.

Capture packets and send packet files along with the time and description of the exception to technical support for analysis.

Run the following command to capture packets:

tcpdump -i eth0 tcp and port 6379 -n -nn -s 74 -w dump.pcap

In Windows, you can also install the Wireshark tool to capture packets.

◻ **NOTE**

For public access, change the port number to **36379**.

Replace the NIC name to the actual one.

----End

4.7 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 4-1 Spring Cloud error information

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.

4.8 What Can I Do If I Fail to Access a DCS Instance Using Its Domain Name Address?

If a client fails to connect to a DCS instance using the domain name address, set the DNS server address of the subnet to the private DNS server address.

For details, see **How Do I Switch to a Private DNS Server?**

4.9 What Should Be Noted When Using Redis for Pub/Sub?

The **Redis official website** describes Pub/Sub in detail. When using Redis for Pub/Sub, note the following:

- 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.

4.10 What Can I Do If Error "Cannot assign requested address" Is Returned When I Access Redis Using connect?

Symptom

Error message "Cannot assign requested address" is returned when you access Redis using **connect**.

Analysis

Applications that encounter this error typically use php-fpm and phpredis. In high-concurrency scenarios, a large number of TCP connections are in the TIME-WAIT state. As a result, the client cannot allocate new ports and the error message will be returned.

Solutions

- Solution 1: Use **pconnect** instead of **connect**.

  Using **pconnect** reduces the number of TCP connections and prevents connections from being re-established for each request, and therefore reduces latency.

  When using **connect**, the code for connecting to Redis is as follows:

  ```
  $redis->connect('${Hostname}',${Port});
  $redis->auth('${Inst_Password}');
  ```

  Replace **connect** with **pconnect**, and the code becomes:

  ```
  $redis->pconnect('${Hostname}', ${Port}, 0, NULL, 0, 0, ['auth' => ['${Inst_Password}']]);
  ```

  📖 **NOTE**

  - Replace the connection parameters in the example with actual values. *${Hostname}*, *${Port}*, and *${Inst_Password}* are the connection address, port number, and password of the Redis instance, respectively.
  - phpredis must be v5.3.0 or later. You are advised to use this **pconnect** initialization mode to avoid NOAUTH errors during disconnection.

- Solution 2: Modify the **tcp_max_tw_buckets** parameter of the ECS where the client is located.

  In this solution, the ports used by TIME-WAIT connections are reused. However, if retransmission occurs between the ECS and the backend service, the connection may fail. Therefore, the **pconnect** solution is recommended.

  a. Connect to the ECS where the client is located

  b. Run the following command to check the **ip_local_port_range** and **tcp_max_tw_buckets** parameters:

     sysctl net.ipv4.tcp_max_tw_buckets net.ipv4.ip_local_port_range

     Information similar to the following is displayed:

     ```
     net.ipv4.tcp_max_tw_buckets = 262144
     net.ipv4.ip_local_port_range = 32768  61000
     ```

      c.   Run the following command to set the **tcp_max_tw_buckets** parameter to a value smaller than the value of **ip_local_port_range**:

         sysctl -w net.ipv4.tcp_max_tw_buckets=10000

Generally, solution 1 is recommended. In special scenarios (for example, the service code involves too many components and is difficult to change), solution 2 can be used to meet high concurrency requirements.

4.11 Connection Pool Selection and Recommended Jedis Parameter Settings

Advantages of the Jedis Connection Pool

The comparison between Lettuce and Jedis is as follows:

- Lettuce
  - Lettuce does not perform connection keepalive detection. If an abnormal connection exists in the connection pool, an error is reported when requests time out.
  - Lettuce does not implement connection pool validation such as **testOnBorrow**. As a result, connections cannot be validated before being used.
- Jedis
  - Jedis implements connection pool validation using **testOnBorrow**, **testWhileIdle**, and **testOnReturn**.

    If **testOnBorrow** is enabled, connection validation is performed when connections are being borrowed, which has the highest reliability but affects the performance (detection is performed before each Redis request).
  - **testWhileIdle** can be used to detect idle connections. If the threshold is set properly, abnormal connections in the connection pool can be removed in time to prevent service errors caused by abnormal connections.
  - If a connection becomes abnormal before the idle connection check, the service that uses the connection may report an error. You can specify the **timeBetweenEvictionRunsMillis** parameter to control the check interval.

Therefore, Jedis has better exception handling and detection capabilities and is more reliable than Lettuce in scenarios where there are connection exceptions and network jitters.

Recommended Jedis Connection Pool Parameter Settings

Table 4-1 Recommended Jedis connection pool parameter settings

Parameter	Description	Recommended Setting
maxTotal	Maximum number of connections	Set this parameter based on the number of HTTP threads of the web container and reserved connections. Assume that the **maxConnections** parameter of the Tomcat Connector is set to **150** and each HTTP request may concurrently send two requests to Redis, you are advised to set this parameter to at least 400 (150 x 2 + 100). **Limit**: The value of **maxTotal** multiplied by the number of client nodes (CCE containers or service VMs) must be less than the maximum number of connections allowed for a single DCS Redis instance. For example, if **maxClients** of a master/standby DCS Redis instance is 10,000 and **maxTotal** of a single client is 500, the maximum number of clients is 20.
maxIdle	Maximum number of idle connections	Set this parameter to the value of **maxTotal**.

Parameter	Description	Recommended Setting
minIdle	Minimum number of idle connections	Generally, you are advised to set this parameter to 1/X of **maxTotal**. For example, the recommended value is **100**. In performance-sensitive scenarios, you can set this parameter to the value of **maxIdle** to prevent the impact caused by frequent connection quantity changes. For example, set this parameter to **400**.
maxWaitMillis	Maximum waiting time for obtaining a connection, in milliseconds	The recommended maximum waiting time for obtaining a connection from the connection pool is the maximum tolerable timeout of a single service minus the timeout for command execution. For example, if the maximum tolerable HTTP timeout is 15s and the timeout of Redis requests is 10s, set this parameter to 5s.
timeout	Command execution timeout, in milliseconds	This parameter indicates the maximum timeout for running a Redis command. Set this parameter based on the service logic. Generally, you are advised to set this timeout to longer than 210 ms to ensure network fault tolerance. For special detection logic or environment exception detection, you can adjust this timeout to seconds.

Parameter	Description	Recommended Setting
minEvictableIdleTimeMillis	Idle connection eviction time, in milliseconds. If a connection is not used for a period longer than this, it will be released.	If you do not want the system to frequently re-establish disconnected connections, set this parameter to a large value (xx minutes) or set this parameter to **–1** and check idle connections periodically.
timeBetweenEviction-RunsMillis	Interval for detecting idle connections, in milliseconds	The value is estimated based on the number of idle connections in the system. For example, if this interval is set to 30s, the system detects connections every 30s. If an abnormal connection is detected within 30s, it will be removed. Set this parameter based on the number of connections. If the number of connections is too large and this interval is too short, request resources will be wasted. If there are hundreds of connections, you are advised to set this parameter to 30s. The value can be dynamically adjusted based on system requirements.
testOnBorrow	Indicates whether to check the connection validity using the **ping** command when borrowing connections from the resource pool. Invalid connections will be removed.	If your service is extremely sensitive to connections and the performance is acceptable, you can set this parameter to **True**. Generally, you are advised to set this parameter to **False** to enable idle connection detection.

Parameter	Description	Recommended Setting
testWhileIdle	Indicates whether to use the **ping** command to monitor the connection validity during idle resource monitoring. Invalid connections will be destroyed.	True
testOnReturn	Indicates whether to check the connection validity using the **ping** command when returning connections to the resource pool. Invalid connections will be removed.	False
maxAttempts	Number of connection retries when JedisCluster is used	Recommended value: 3–5. Default value: **5**. Set this parameter based on the maximum timeout intervals of service APIs and a single request. The maximum value is **10**. If the value exceeds **10**, the processing time of a single request is too long, blocking other requests.

4.12 What Can I Do If a Lettuce 6.x Client Is Incompatible with My DCS Instance?

Symptom

When a Lettuce 6.x client is connected to a Proxy Cluster DCS Redis 4.x/5.x instance, the error message "NOAUTH Authentication required" is displayed.

Figure 4-2 Error message example

```
[2022-01-04 18:33:35.219] [lettuce-nioEventLoop-4-1] [DEBUG] [io.lettuce.core.AbstractRedisClient:?] - Connecting to Redis at 192.168.xxx.xxx:6379, initialization
java.util.concurrent.CompletionException: io.lettuce.core.RedisCommandExecutionException: NOAUTH Authentication required.
    at java.util.concurrent.CompletableFuture.encodeThrowable(CompletableFuture.java:292)
    at java.util.concurrent.CompletableFuture.completeThrowable(CompletableFuture.java:308)
```

Analysis

In Lettuce 6.x and later versions, the **HELLO** command of RESP3 (introduced in Redis 6.x) is used to determine the version adaptation. Instances of earlier versions

that do not support the **HELLO** command may encounter compatibility issues. For these instances, you can specify the RESP2 mode (compatible with Redis versions 4 and 5) in Lettuce.

Solution

Add the following code to use RESP2 protocol for accessing Redis:

```
package com.chinaroad.parking.config;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.protocol.ProtocolVersion;
import org.springframework.boot.autoconfigure.data.redis.LettuceClientConfigurationBuilderCustomizer;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;

@Configuration
public class SpringConfig implements LettuceClientConfigurationBuilderCustomizer {

    @Override
    public void customize(LettuceClientConfiguration.LettuceClientConfigurationBuilder
clientConfigurationBuilder) {
        // manually specifying RESP2
        clientConfigurationBuilder.clientOptions(ClientOptions.builder()
            .protocolVersion(ProtocolVersion.RESP2)
            .build());
    }

}
```

4.13 Should I Use a Domain Name or an IP Address to Connect to a DCS Redis Instance?

- Single-node, read/write splitting, and Proxy Cluster:

  Each instance has only one IP address and one domain name address. The addresses remain unchanged before and after master/standby switchover. You can use either address to connect to the instance.

- Master/Standby:

  Each instance has one IP address and two domain name addresses. One of the domain name addresses is used only for processing read requests. The addresses remain unchanged after master/standby switchover. You can use any address to connect to the instance.

  When you use a domain name address, distinguish between read and write requests. If you use **Connection Address** or **IP Address**, functions are not affected. If you use **Read-only Address**, only read requests are processed. You are advised to use read/write splitting instances if you have read/write splitting requirements.

- Redis Cluster:

  A Redis Cluster instance has multiple pairs of master and replica IP addresses and one domain name address. You can use any address to connect to the instance.

  The connected node sends requests to the correct node. All nodes in the cluster can receive requests. **Configure multiple or all IP addresses** to prevent single points of failure.

> 📖 **NOTE**

- Domain names cannot be resolved across regions. If the client and the DCS Redis instance are not in the same region, the instance cannot be accessed using its domain name address. You can manually map the domain name to the IP address in the **hosts** file or access the instance using its IP address. For details, see **Restrictions**.
- For details about how to connect to an instance, see **Accessing a DCS Redis Instance**.

4.14 Is the Read-only Address of a Master/Standby Instance Connected to the Master or Standby Node?

A master/standby DCS Redis 4.0/5.0/6.0 instance has a **Connection Address** and a **Read-only Address**. The connection address is used to connect to the master node of the instance, and the read-only address is used to connect to the standby node of the instance.

For details, see **Architecture of Master/Standby DCS Redis 4.0/5.0/6.0 Basic Instances**.

Figure 4-3 Instance addresses

By default, the client reads and writes data on the master node. Data is synchronized to the standby node. To implement read/write splitting, ensure that your client can distinguish between read and write requests. The client directs write requests to the read/write domain name and read requests to the read-only domain name.

5 **Redis Usage**

5.1 Can I Change the AZ for an Instance?

No.

To use a different AZ, create another instance in the desired AZ, and then migrate data and switch IP addresses. For details, see **IP Switching**.

5.2 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. To use a different set of VPC and subnet, create another instance with a desired set of VPC and subnet and then migrate data as required.

5.3 Can I Customize or Change the Port for Accessing a DCS Instance?

Customizing a Port

When creating a DCS Redis 4.0, 5.0, or 6.0 instance, you can enter a port number for **IP Address**. If you do not specify a port, the default port 6379 is used.

Changing the Port

After a DCS Redis 4.0, 5.0, or 6.0 instance is created, you can change its port.

1. In the navigation pane of the DCS console, choose **Cache Manager**.
2. Click a DCS Redis instance.
3. In the **Connection** area, click 🖉 next to **Connection Address**.

> **NOTICE**
>
> After the port is changed, all connections to the Redis instance are interrupted, and services are connected to the new port.

5.4 Can I Modify the Connection Addresses for Accessing a DCS Instance?

After a DCS instance is created, its IP address and domain name for intra-VPC access cannot be modified.

To use a different IP address, you must create a new instance and manually specify an IP address. After the instance is created, migrate the data from the old instance to the new instance.

5.5 Why Does It Take a Long Time to Start a Cluster DCS Instance?

Possible cause: When a cluster instance is started, status and data are synchronized between the nodes of the instance. If a large amount of data is continuously written into the instance before the synchronization is complete, the synchronization will be prolonged and the instance remains in the **Starting** state. After the synchronization is complete, the instance enters the **Running** state.

Solution: Start writing data to an instance only after the instance has been started.

5.6 What Should I Do If an Error Occurs in redis_exporter?

Start redis_exporter using the CLI. Based on the output, check for errors and troubleshoot accordingly.

```
[root@ecs-swk /]./redis_exporter -redis.addr 192.168.0.23:6379
INFO[0000] Redis Metrics Exporter V0.15.0   build date:2018-01-19-04:08:01 sha1:
a0d9ec4704b4d35cd08544d395038f417716a03a
  Go:go1.9.2
INFO[0000] Providing metrics at :9121/metrics
INFO[0000] Connecting to redis hosts: []string{192.168.0.23:6379}
INFO[0000] Using alias:[]string{""}
```

5.7 Does DCS for Redis Provide Backend Management Software?

No. To query Redis configurations and usage information, use redis-cli. If you want to monitor DCS Redis instance metrics, go to the Cloud Eye console or perform the following operations.

Procedure

Step 1 Log in to the management console, and choose **Application** > **Distributed Cache Service** in the service list.

Step 2 Click [icon] in the upper left corner of the management console and select the region where your instance is located.

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

Step 4 Click the 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

5.8 Can I Recover Deleted Data of a DCS Instance?

If you have backed up the DCS instance, you can restore its data from the backup. However, the restoration will overwrite the data written in before the restoration.

You can restore backup data to a master/standby, cluster, or read/write splitting instance through **Backups & Restorations** on the DCS console. For details, see **Restoring a DCS Instance**.

If a DCS instance is deleted, the instance data and its backup will also be deleted. Before deleting an instance, you can download the backup files of the instance for permanent local storage and can also migrate them to a new instance if you need to restore the data. For details about how to download the backup data, see **How Do I Export DCS Redis Instance Data?**

5.9 How Do I Check Redis Memory Usage?

Currently, DCS for Redis provides the following memory-related metrics: For details about how to check monitoring metrics, see **Viewing Metrics**.

Table 5-1 DCS Redis instance metrics

Metric ID	Metric Name	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Only Redis 4.0 and later Dimension: dcs_instance_id	1 minute
used_memory_dataset_perc	Used Memory Dataset Ratio	Percentage of data memory that Redis has used to the total used memory Unit: %	0–100%	Monitored object: DCS Redis instance Only Redis 4.0 and later Dimension: dcs_instance_id	1 minute

Metric ID	Metric Name	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: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: 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: 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: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute

5.10 Why Is the Capacity or Performance of a Shard of a Redis Cluster Instance Overloaded When That of the Instance Is Still Below the Bottleneck?

Redis Cluster uses a special data sharding method. **Every key is part of a hash slot, which is held by a node in the cluster.** To compute what is the hash slot of a given key:

1. Take the CRC16 of the key modulo 16384.

2. Based on the mapping between hash slots and shards, connections are redirected to the right node for data read and write operations.

Therefore, keys are not evenly distributed to each shard of an instance. If a shard contains a big key or a hot key, the capacity or performance of the shard will be overloaded, but the load on other shards is still low. As a result, the capacity or performance bottleneck of the entire instance is not reached.

5.11 Why Does an OOM Error Occur During a Redis Connection?

Symptom

"Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'" is returned during a Redis connection.

Fault Locating

An out-of-memory (OOM) error indicates that the maximum memory is exceeded. In the error information, the **maxmemory** parameter indicates the maximum memory configured on the Redis server.

If the memory usage of the Redis instance is less than 100%, the memory of the node where data is written may have reached the maximum limit. Connect to each node in the cluster by running **redis-cli -h <redis_ip> -p 6379 -a <redis_password> -c --bigkeys**. When connecting to a replica node, run the **READONLY** command before running the **bigkeys** command.

5.12 What Clients Can I Use for Redis Cluster in Different Programming Languages?

The following table compares Redis Cluster and Proxy Cluster in DCS.

Table 5-2 Comparing Redis Cluster and Proxy Cluster

Item	Redis Cluster	Proxy Cluster
Redis compatibility	High	Medium
Client compatibility	Medium (The cluster mode must be enabled on the client.)	High
Costs	High	Medium
Latency	Low	Medium
Read/write splitting	Native support (client SDK configuration)	Implemented by using proxies
Performance	High	Medium

Redis Cluster does not use proxies, and therefore delivers lower latency and higher performance. However, Redis Cluster instances are based on the open-source Redis Cluster protocol, so their client compatibility is poorer than that of Proxy Cluster instances.

The following table lists clients that can be used for Redis Cluster.

Table 5-3 Clients that can be used for Redis Cluster

Language	Client	Reference Document
Java	Jedis	**https://github.com/ xetorthio/jedis#jedis- cluster**
Java	Lettuce	**https://github.com/ lettuce-io/lettuce-core/ wiki/Redis-Cluster**
PHP	php redis	**https://github.com/ phpredis/ phpredis#readme**
Go	Go Redis	Redis Cluster: **https:// pkg.go.dev/ github.com/go-redis/ redis/ v8#NewClusterClient** Proxy Cluster, single-node, or master/standby: **https://pkg.go.dev/ github.com/go-redis/ redis/v8#NewClient**

Language	Client	Reference Document
Python	redis-py-cluster	**https://github.com/ Grokzen/redis-py- cluster#usage-example**
C	hiredis-vip	**https://github.com/ vipshop/hiredis-vip? _ga=2.64990636.268662 337.1603553558-97776 0105.1588733325**
C++	redis-plus-plus	**https://github.com/ sewenew/redis-plus- plus? _ga=2.64990636.268662 337.1603553558-97776 0105.1588733325#redis -cluster**
Node.js	node-redis io-redis	**https://github.com/ NodeRedis/node-redis** **https://github.com/ luin/ioredis**

To view all Redis clients, see **https://redis.io/clients**.

5.13 Why Do I Need to Configure Timeout for Redis Cluster?

If timeout is not configured, connections will fail.

When you connect to a Redis Cluster instance using Spring Boot and Lettuce, connect to all cluster nodes, including faulty replicas.

- If timeout is not configured, minute-level blocking (120s in earlier Lettuce versions and 60s in the new version) may occur when there is a faulty replica, as shown in the following figure.
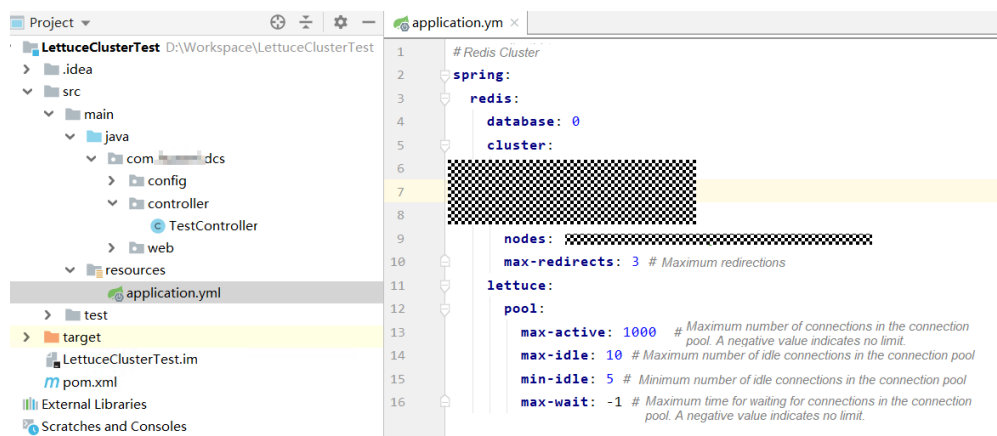
The end-to-end service access time may reach the maximum timeout, as shown in the following figure.


```
[root@ecs-a776 ~]# time curl -X get http://1 .0.0.1:8080/test/evalsha
false
real    2m0.632s
user    0m0.003s
sys     0m0.004s
```

- After the **timeout** parameter is configured on the client, the timeout on the replica will be greatly shortened. You can adjust the timeout based on the service requirements. Assume that the configuration is as follows.
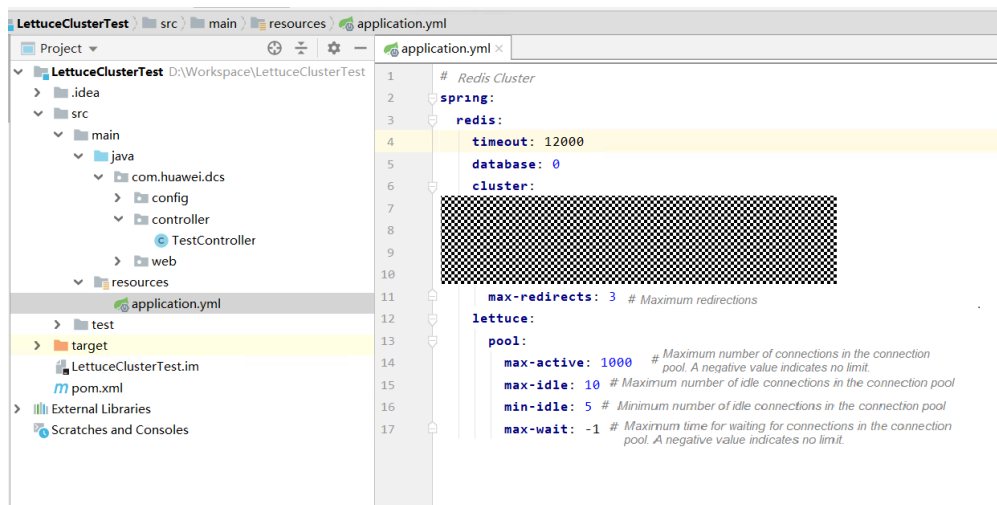
The following figure shows the end-to-end service access time after the configuration is complete.


```
[root@ecs-a776 ~]# time curl -X get http://1  .0.0.1:8080/test/evalsha
false
real    0m12.627s
user    0m0.000s
sys     0m0.004s
```

If the **timeout** parameter is not configured and there is a faulty node, client connections will be blocked.

Configure the timeout based on what the service can tolerate. For example, if you need to request Redis twice in an HTTP request and the maximum timeout of an HTTP request is 10s, it is recommended that you set the timeout in Redis to 5s. This prevents service interruption if faults occur due to a long timeout duration or no timeout duration.

5.14 Why Am I Seeing a Timeout Error When Reading Data from Redis?

Symptom

When you read data from Redis, timeout error "redis server response timeout (3000ms) occurred after 3 retry attempts" is returned.

Troubleshooting

1. Increase the timeout based on the error information.

2. Check whether the error is returned after an operation is performed on a big key. It is recommended that the key size be no greater than 10 KB.

   Redis limits the size of each String value to 512 MB. In actual development, keep the size within 10 KB. Otherwise, the CPU and NIC will be heavily loaded.

   Keep the number of Hashes, Lists, Sets, or Zsets within 5000.

   Theoretically, the number of elements in each HashSet should be fewer than 2^32.

3. Increase the value of parameter **PingConnectionInterval** based on the error information.

5.15 Explaining and Using Hash Tags

Hash Tag Design

Multi-key operations, such as those using the **MSET** command or Lua scripts, are atomic. All specified keys are executed at the same time. However, in a cluster, each key is hashed to a given shard, and multi-key operations are no longer atomic. The keys may be allocated to different slots. As a result, some keys are updated, while others are not. If there is a hash tag, the cluster determines which slot to allocate a key based on the hash tag. Keys with the same hash tag are allocated to the same slot.

Using Hash Tags

Only the content between the first left brace ({) and the following first right brace (}) is hashed.

For example:

- In keys **{user1000}.following** and **{user1000}.followers**, there is only one pair of braces. **user1000** will be hashed.

- In key **foo{}{bar}**, there is no content between the first { and the first }. The whole key **foo{}{bar}** will be hashed as usual.

- In key **foo{{bar}}zap**, **{bar** (the content between the first { and the first }) is hashed.

- In key **foo{bar}{zap}**, **bar** is hashed because it is between the first pair of { and }.

Hash Tag Example

When the following operation is performed:

```
EVAL "redis.call('set',KEYS[1],ARGV[1]) redis.call('set',KEYS[2],ARGV[2])" 2 key1 key2 value1 value2
```

The following error is displayed:

ERR 'key1' and 'key2' not in the same slot

You can use a hash tag to solve this issue:

```
EVAL "redis.call('set',KEYS[1],ARGV[1]) redis.call('set',KEYS[2],ARGV[2])" 2 {user}key1 {user}key2 value1
value2
```

5.16 Why Does a Key Disappear in Redis?

Normally, Redis keys do not disappear. If a key is missing, it may have expired, been evicted, or been deleted.

Perform the following checks one by one:

1. Check whether the key has expired.
2. View the monitoring information and check whether eviction was triggered.
3. Run the **INFO** command on the server side to check whether the key has been deleted.

5.17 Will Cached Data Be Retained After an Instance Is Restarted?

After a single-node DCS instance is restarted, data in the instance is deleted.

Master/Standby, read/write splitting, and cluster instances (except single-replica clusters) support AOF persistence by default. Data is retained after these instances are restarted.

If AOF persistence is disabled (**appendonly** is set to **no**), data is deleted after the instances are restarted.

5.18 How Do I Know Whether an Instance Is Single-DB or Multi-DB?

Single-node, read/write splitting, and master/standby: multi-DB (256 DBs, numbered from 0 to 255)

Proxy Cluster: single-DB by default. Multi-DB can be enabled. For details, see **What Are the Constraints on Implementing Multiple Databases on a Proxy Cluster Instance?**

Redis Cluster: single-DB. Multi-DB is not supported.

You can connect to a DCS Redis 4.0 or later instance on the console to check whether it is multi-DB.

Figure 5-1 Connecting to Redis
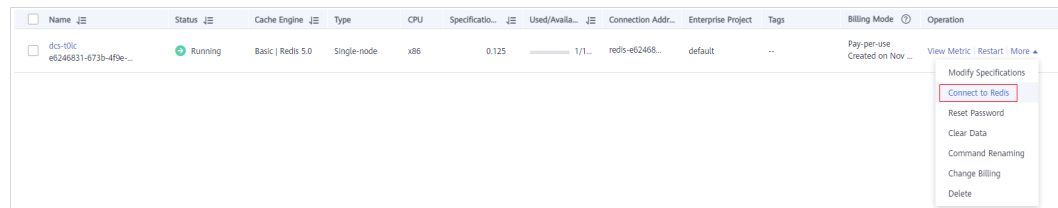
Figure 5-2 Viewing databases
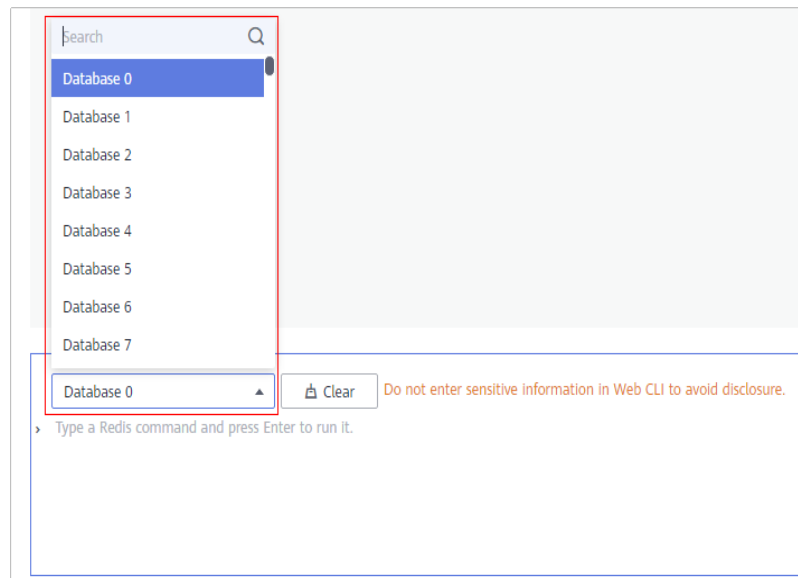
5.19 What Are the Constraints on Implementing Multi-DB on a Proxy Cluster Instance?

Note the following constraints when you consider implementing multi-DB:

- **Usage constraints:**

  a. The **SWAPDB** command does not support multi-DB.

  b. The **INFO KEYSPACE** command does not return data of multi-DB.

  c. To query the total number of keys in each database, use the customized **dbstats** command. CPU usage will surge on the node executing this command.

  d. LUA scripts do not support multi-DB.

  e. The **RANDOMKEY** command does not support multi-DB.

  f. The **SELECT** command cannot be embedded in transactions.

  g. **PUBLISH** cannot be used in Lua scripts.

  h. The database number ranges from 0 to 255.

- **Performance constraints**

  a. The **FLUSHDB** command deletes keys one by one, which takes a long time and is slower than the open-source native implementation. The

execution speed of the **FLUSHDB** command is the same as that of the **SCAN** command (which should be tested by the customer).

    b. The **DBSIZE** command is time-consuming. Do not use it in the code.

    c. If multi-DB is used, the performance of the **KEYS** and **SCAN** commands deteriorates by up to 50%.

- **Other constraints**:

  The backend storage rewrites keys based on certain rules. Keys in the exported RDB file are not the original keys. However, the access through the Redis protocol is not affected.

Procedure for Enabling or Disabling Multi-DB

By default, a Proxy Cluster instance is not enabled with multi-DB, but you can do as follows to enable multi-DB.

Step 1 Log in to the DCS console.

Step 2 Connect to the instance and run the **FLUSHALL** command to clear the instance data.

> **NOTE**
>
> Before enabling or disabling multi-DB, ensure that all instance data is cleared and no data is being written.

Step 3 On the **Cache Manager** page of the DCS console, click the desired DCS instance.

Step 4 Choose **Instance Configuration** > **Parameters**.

Step 5 To enable multi-DB, click **Modify** in the row that contains parameter **multi-db**, and change its value to **yes**.

To disable multi-DB, change the value to **no**.

Step 6 Click **Save** and then confirm the modification. The instance does not need to be restarted.

maxmemory-policy ⓘ	volatile-lru	volatile-lru,allkeys-lru,volatile-lfu,allkeys-lfu,volatile-random,allkeys-random,volatile-t...	volatile-lru	Modify
multi-db ⓘ	no	no,yes	no	Modify
multi-db-keys-scan-enabled ⓘ	no	no,yes	no	Modify

----End

5.20 How Do I Create a Multi-DB Proxy Cluster Instance?

When you create a Proxy Cluster instance, there is only one database by default. This section describes how to create a Proxy Cluster instance with multiple databases.

> **NOTE**
>
> Before getting started, learn about **the constraints on implementing multi-DB**.

Step 1 Log in to the management console, and choose **Application** > **Distributed Cache Service** in the service list.

Step 2 Click ⊙ in the upper left corner to select a region.

Step 3 In the navigation pane, choose **Parameter Templates**.

Step 4 In the row that contains the template with the desired cache engine version and instance type (Proxy Cluster), click **Customize**.

Step 5 Set **multi-db** to **yes**.

Step 6 Enter a new template name and click **OK**. The custom template is created successfully.

Step 7 In the navigation pane, choose **Cache Manager**. Then click Buy DCS Instance to create a Proxy Cluster instance.

Set **Parameter Configuration** to **Use custom template** and select the custom template created in the preceding step.

After the instance is created, connect to it to check whether it has multiple databases.

----End

6 Instance Scaling and Upgrade

6.1 Can I Upgrade Version for a DCS Redis Instance, for Example, from Redis 4.0 to Redis 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.

If your service requires the features of higher Redis versions, create a new 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 the **Data Migration Guide**.

6.2 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.

6.3 Are Instances Stopped or Restarted During Specification Modification?

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

6.4 What DCS Instance Type Changes Are Supported?

Table 6-1 Instance type change options supported by different DCS instances

Version	Supported Type Change	Precautions
Redis 4.0/5.0	From master/standby or read/write splitting to Proxy Cluster From Proxy Cluster to master/standby or read/write splitting	1. Before changing the instance type to Proxy Cluster, evaluate the impact on services. For details, see **What Are the Constraints on Implementing Multi-DB on a Proxy Cluster Instance?** and **Command Restrictions**. 2. Memory usage must be less than 70% of the maximum memory of the new flavor. 3. Some keys may be evicted if the current memory usage exceeds 90% of the total. 4. After the change, **create alarm rules** again for the instance. 5. For instances that are currently master/standby, ensure that their read-only IP address or domain name is not used by your application. 6. If your application cannot reconnect to Redis or handle exceptions, you may need to restart the application after the change. 7. Modify instance specifications during off-peak hours. An instance is temporarily interrupted and remains read-only for about 1 minute during the specification change.

For details about the commands supported by different types of instances, see **Command Compatibility**.

Any instance type changes not listed in the preceding table are not supported. To modify specifications while changing the instance type, see **IP Switching**.

To check whether you can change the instance type of an instance, see the parameters displayed on the **Modify Specifications** page on the DCS console.

6.5 Are Services Interrupted During Specification Modification?

Modify instance specifications during off-peak hours.

If the modification failed in peak hours (for example, when memory or CPU usage is over 90% or write traffic surges), try again during off-peak hours.

The following table describes the impact of specification modification.

Change of the Instance Type

Table 6-2 Instance type change options supported by different DCS instances

Version	Supported Type Change	Precautions
Redis 4.0/5.0	From master/standby or read/write splitting to Proxy Cluster From Proxy Cluster to master/standby or read/write splitting	1. Before changing the instance type to Proxy Cluster, evaluate the impact on services. For details, see **What Are the Constraints on Implementing Multi-DB on a Proxy Cluster Instance?** and **Command Restrictions**. 2. Memory usage must be less than 70% of the maximum memory of the new flavor. 3. Some keys may be evicted if the current memory usage exceeds 90% of the total. 4. After the change, **create alarm rules** again for the instance. 5. For instances that are currently master/standby, ensure that their read-only IP address or domain name is not used by your application. 6. If your application cannot reconnect to Redis or handle exceptions, you may need to restart the application after the change. 7. Modify instance specifications during off-peak hours. An instance is temporarily interrupted and remains read-only for about 1 minute during the specification change.

Any instance type changes not listed in the preceding table are not supported. To modify specifications while changing the instance type, see **IP Switching**.

For details about the commands supported by different types of instances, see **Command Compatibility**.

Scaling

- **Scaling options**

  Table 6-3 Scaling options supported by different instances

Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster	Read/Write Splitting
Redis 4.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/down, out/in, and replica quantity change	Scaling up/down and out/in	Scaling up/down and replica quantity change
Redis 5.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/down, out/in, and replica quantity change	Scaling up/down and out/in	Scaling up/down and replica quantity change
Redis 6.0	Scaling up/down	Scaling up/down	-	-	-

- **Impact of scaling**

Table 6-4 Impact of scaling

Instance Type	Scaling Type	Impact
Single-node , read/write splitting, and master/standby	Scaling up/down	• During scaling up, a DCS Redis 4.0/5.0/6.0 instance will be disconnected for several seconds and remain read-only for about 1 minute. During scaling down, connections will not be interrupted. • For scaling up, only the memory of the instance is expanded. The CPU processing capability is not improved. • Single-node DCS instances do not support data persistence. Scaling may compromise data reliability. After scaling, check whether the data is complete and import data if required. If there is important data, use a migration tool to migrate the data to other instances for backup. • For master/standby and read/write splitting instances, backup records created before scale-down cannot be used after scale-down. If necessary, download the backup file in advance or back up the data again after scale-down.

Instance Type	Scaling Type	Impact

Proxy Cluster and Redis Cluster	Scaling up/down	• Scaling out by adding shards:
		  – **Scaling out does not interrupt connections but will occupy CPU resources, decreasing performance by up to 20%.**
		  – If the shard quantity increases, new Redis Server nodes are added, and data is automatically balanced to the new nodes, increasing the access latency.
		• Scaling in by reducing shards:
		  – If the shard quantity decreases, nodes will be deleted. **Before scaling in a Redis Cluster instance, ensure that the deleted nodes are not directly referenced in your application, to prevent service access exceptions.**
		  – **Nodes will be deleted**, and connections will be interrupted. If your application cannot reconnect to Redis or handle exceptions, you may need to restart the application after scaling.
		• Scaling up by shard size without changing the shard quantity: Currently unavailable.
		• Scaling down by reducing the shard size without changing the shard quantity has no impact.
		• To scale down an instance, ensure that the used memory of each node is less than 70% of the maximum memory per node of the new flavor.
		• **The flavor changing operation may involve data migration, and the latency may increase. For a Redis Cluster instance, ensure that the client can process the MOVED and ASK commands. Otherwise, the request will fail.**
		• If the memory becomes full during scaling due to a large amount of data being written, scaling will fail.
		• Before scaling, **check for big keys through Cache Analysis**. Redis has a limit on key migration. If the instance has any single key greater than 512 MB, scaling will fail when big key migration between nodes times out. The bigger the key, the more likely the migration will fail.
		• **Before scaling a Redis Cluster instance, ensure that automated cluster topology refresh is enabled.** If it is disabled, you will need to restart the client after scaling. For details about how to enable automated refresh if you use Lettuce, see **an example of using Lettuce to connect to a Redis Cluster instance**.

Instance Type	Scaling Type	Impact
		• Backup records created before scaling cannot be used. If necessary, download the backup file in advance or back up the data again after scaling.
Master/ standby, read/ write splitting, and Redis Cluster instances	Scaling out/in (replica quantity change)	• Before adding or removing replicas for a Redis Cluster instance, ensure that automated cluster topology refresh is enabled. If it is disabled, you will need to restart the client after scaling. For details about how to enable automated refresh if you use Lettuce, see **an example of using Lettuce to connect to a Redis Cluster instance**. • Deleting replicas interrupts connections. If your application cannot reconnect to Redis or handle exceptions, you may need to restart the application after scaling. Adding replicas does not interrupt connections. • If the number of replicas is already the minimum supported by the instance, you can no longer delete replicas.

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

- Check whether other tasks are running.

  Specifications of a DCS instance cannot be modified if another task of the instance is still running. For example, you cannot delete or scale up an instance while it is being restarted. Likewise, you cannot delete an instance while it is being scaled up.

  If the specification modification fails, try again later. If it fails again, contact technical support.

- When changing a master/standby instance to the Proxy Cluster type, check whether data exists in DBs other than DB0. Specification modification will fail if a DB other than DB0 contains data.

  A master/standby instance can be changed to the Proxy Cluster type when data exists only in DB0.

6.7 How Do I Reduce the Capacity of a DCS Instance?

Table 6-5 lists scaling options supported by different DCS instances.

Table 6-5 Scaling options supported by different instances

Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster	Read/Write Splitting
Redis 4.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/down, out/in, and replica quantity change	Scaling up/down and out/in	Scaling up/down and replica quantity change
Redis 5.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/down, out/in, and replica quantity change	Scaling up/down and out/in	Scaling up/down and replica quantity change
Redis 6.0	Scaling up/down	Scaling up/down	-	-	-

For details about how to change the capacity, see **Modifying Specifications**.

6.8 How Do I Add Shards to a Cluster DCS Redis Instance Without Changing the Memory?

After a Proxy Cluster or Redis Cluster instance is created, you can reduce the capacity of each shard and add more shards without changing the total memory.

For example, if an 8 GB instance has 4 shards and each shard is 2 GB, you can reduce the shard size to 1 GB and increase the shard quantity to 8.

📖 **NOTE**

A shard size of 1 GB cannot be changed.

Procedure

Step 1 Log in to the management console, and choose **Application** > **Distributed Cache Service** in the service list.

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

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

Step 4 Choose **More** > **Modify Specifications** in the row containing the desired DCS instance.

Step 5 On the displayed **Modify Instance Specifications** page, specify **Shard Size** and **Instance Specification**.

Figure 6-1 Selecting a shard size
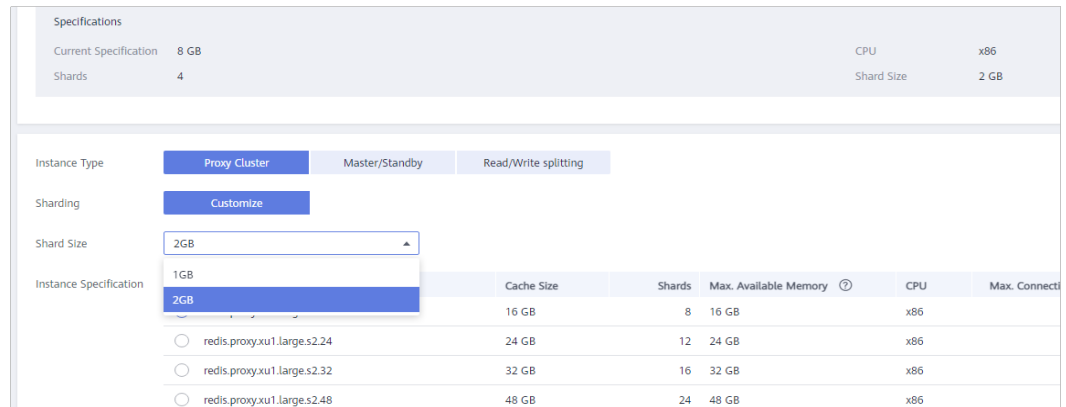
Step 6 Click **Next**, confirm the details, and click **Submit**.

The modification takes about 5 to 30 minutes to complete. After the modification is successful, the instance status changes to **Running**.

----End

6.9 How Do I Handle an Error When I Use Lettuce to Connect to a Redis Cluster Instance After Specification Modification?

Symptom

If the shard quantity changes during specification modification of a Redis Cluster instance, some slots are migrated to new shards. The following error occurs when you use Lettuce to connect to the instance.

Figure 6-2 Error

For details, see **Connection to X not allowed. This connection point is not known in the cluster view**.

Analysis

Specification modification process of a Redis Cluster instance:

After being started, the client obtains the cluster node topology by using the **Cluster Nodes** command based on RESP2, and maintains the topology in its in-memory data structure.

For data access, the client uses the CRC16 algorithm to calculate the hash slot of a key, and automatically routes requests based on the topology and slot information stored in the memory.

If the number of shards changes during scaling, the topology and slot mapping changes. In this case, the client needs to automatically update the topology. Otherwise, the request route may fail or the route location may be incorrect. As a result, an error is reported during client connection.

For example, when the number of shards in a Redis Cluster instance changes from three to six, the topology and slot mapping changes as shown in the following figures.

Figure 6-3 A Redis Cluster instance before scaling up
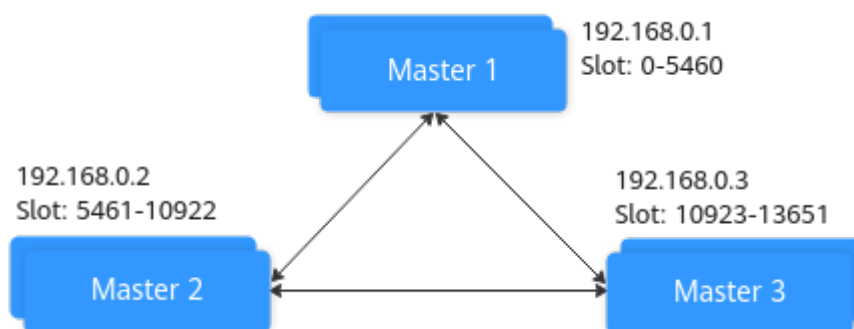
Figure 6-4 A Redis Cluster instance after scaling up
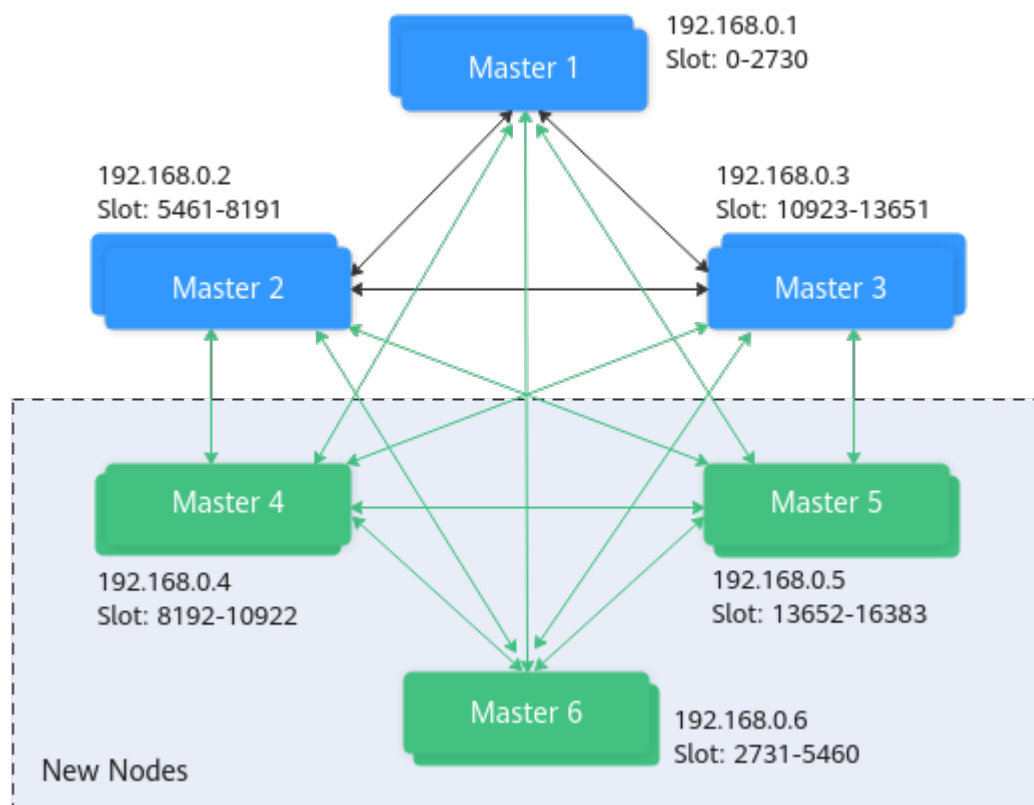
Solutions

Solution 1 (Recommended)

Enable automated topology refresh.

```
ClusterTopologyRefreshOptions topologyRefreshOptions = ClusterTopologyRefreshOptions.builder()
        // Periodic refresh: every time milliseconds.
        .enablePeriodicRefresh(Duration.ofMillis(time))
      // Triggers of adaptive refresh: MOVED redirection, ASK redirection, reconnection, unknown node
(since 5.1), and slot not in any of the current shards (since 5.2).
        .enableAllAdaptiveRefreshTriggers()
        .build();
```

For details, see **an example of using Lettuce to connect to a Redis Cluster instance**.

☐ NOTE

> If you use Lettuce to connect to a Redis Cluster instance and automated refresh is not
> enabled, you need to restart the client after specification modification.

Solution 2

Disable validation of cluster node membership.

```
ClusterClientOptions clusterClientOptions = ClusterClientOptions.builder()
      .validateClusterNodeMembership(false)
      .build();
```

If **validateClusterNodeMembership** is **true**, check whether the current connection address is in the cluster topology obtained through **CLUSTER NODES**, before connecting to the cluster. If it is not in the topology, the error occurs.

☐ NOTE

> Impact of disabling validation of cluster node membership:
>
> - Lack of security breach detection.
> - If automated topology refresh is disabled, a MOVED redirection request may be
>   generated after the Redis Cluster specifications are changed and the shard quantity
>   increases. Redirection increases the network load of the cluster and the time required to
>   process a single request. If the shard quantity decreases, deleted shards cannot be
>   connected.

6.10 Can I Expand a Single Shard of a Cluster Instance (Scale-Up)?

No. You can only add more shards to expand the instance capacity.

If you want to use a larger size in each shard of a Proxy Cluster instance, change the instance to the master/standby type, and then change it back to Proxy Cluster with the desired shard size. Before changing the instance type to Proxy Cluster, evaluate the impact on services. For details, see **What Are the Constraints on Implementing Multi-DB on a Proxy Cluster Instance?**

7 Data Backup, Export, and Migration

7.1 Can I Migrate Data from a Lower Redis Version to a Higher One?

Yes. Redis is backward compatible.

The version of the source Redis (DCS, self-built, or another cloud) can be earlier than or the same as the target DCS instance.

7.2 What Should I Consider When Transferring or Operating Data Between Different OSs?

Convert the format of a data file before importing the file.

Run the following command to convert the format of a file in the Windows OS to that in the Unix-like OS:

dos2unix *{filename}*

Run the following command to convert the format of a file in the Unix-like OS to that in the Windows OS:

unix2dos *{filename}*

7.3 Can I Migrate Data from a Multi-DB Source Redis Instance to a Cluster DCS Redis Instance?

A total of 256 DBs (DB 0 to DB 255) can be configured for a single-node, read/write splitting, or master/standby DCS instance.

- If the target is a Redis Cluster instance (a Redis Cluster instance has only one DB):

  Solutions:

  a. Combine different DBs in the source Redis instance into one DB.

  b. Apply for multiple DCS instances.

After the migration, the instance connection address and DB IDs change. In this case, modify the configurations for your services.

- If the target is a Proxy Cluster instance:

By default, the multi-DB function is disabled for Proxy Cluster instances and only one DB (DB0) is available. To use more than one DB, enable multi-DB by referring to **Enabling Multi-DB** before migration.

7.4 What Are the Constraints and Precautions for Migrating Redis Data to a Cluster Instance?

- Proxy Cluster instances

Proxy Cluster instances are used in the same way that you use single-node or master/standby instances. However, only one DB is configured for a Proxy Cluster instance by default, and the **SELECT** command is not supported. When data files are imported in batches, an error message will be displayed and ignored if the **SELECT** command exists. Then, the remaining data will continue to be imported.

Example:

DB 0 and DB 2 in the source Redis instance contain data, and the generated AOF or RDB file contains these two DBs.

When the source Redis data is imported into a Proxy Cluster DCS instance, the **SELECT 2** command will be ignored, and then data in DB 2 in the source Redis instance will be imported.

Note that:

- – If different DBs in the source Redis instance contain the same keys, values of keys in the DB with the largest ID will overwrite those in the other DBs.

- – If the source Redis instance contains multiple DBs, data is stored in the same DB after being migrated to a cluster DCS instance, and the **SELECT** command is not supported. In this case, modify the configurations for your services.

- Redis Cluster instances

Only one DB is configured for a Redis Cluster instance. Data is migrated to a Redis Cluster instance in a different way from other types of instances. Nodes in the shards of a Redis Cluster must be connected separately through clients. Data is imported to the nodes separately. Run the following command to query the IP addresses of the cluster nodes:

redis-cli -h *{Redis Cluster IP}* **-p 6379 -a** *{password}* **cluster nodes**

In the returned list of IP addresses, record the ones marked by "master".

7.5 What Should I Consider for Online Migration?

- Network

Before online migration, ensure that the network configured for the migration task is connected to source and target Redis instances.

- Tool

  Use the online migration function provided on the DCS console.

- Data integrity

  If you suspend your services for data migration, check the data volume and main keys after the migration.

  If you do not suspend your services, migrate data incrementally.

- Impact of capacity expansion of the source instance

  During online migration, expanding the source instance's capacity may affect the migration or customer data. If the memory of the source instance becomes insufficient during the migration, stop the migration task and then expand the capacity.

- Timing

  Migration should take place during off-peak hours.

- Version restrictions

  You can migrate data from an earlier version to a later version, and vice versa, but you need to check whether the target instance supports the commands used in your service systems.

- Multi-DB

  If both the target and source are Proxy Cluster DCS instances, ensure that the **multi-db** parameter settings of the two are the same. Otherwise, the migration will fail.

7.6 Can I Perform Online Migration Without Any Service Interruption?

Yes. You can use the application dual-write mode. In this mode, during data migration, data is still read from the source Redis instance, and operations such as adding, deleting, and modifying data are also performed on the DCS Redis instance.

After maintaining the preceding mode for a period of time (waiting for a large amount of data to be deleted after expiration), migrate the cached data from your service systems to DCS. If service system migration to the cloud service is also involved, deploy your service systems before migrating your cached data.

This mode is not recommended for the following reasons:

1. Stable and quick network access cannot be ensured. If the source Redis instance is not deployed on DCS, access DCS over a public network, which is inefficient.
2. Modify the code to implement concurrent writing of two sets of data.
3. The data eviction policy varies depending on the source Redis instance. It may take a long time to complete data migration and it is difficult to ensure data integrity.

7.7 What If "Disconnecting timedout slave" and "overcoming of output buffer limits" Are Reported on the Source Instance During Online Migration?

The following error messages may be displayed during online migration:

- "Disconnecting timedout slave" is reported on the source instance, as shown in the following figure:

Solution: Set the **repl-timeout** parameter of the source Redis instance to 300s.

- "overcoming of output buffer limits" is reported on the source instance, as shown in the following figure:

Solution: Set the **client-output-buffer-limit** parameter of the source Redis instance to 20% of the maximum memory of the instance.

7.8 How Do I Export DCS Redis Instance Data?

- Master/standby and cluster instances:

  These instances support backups. Perform the following operations to export data:

  a. On the **Backups and Restorations** page, view the backup tasks.

  b. If there is no backup, create a backup and download the backup file as prompted.

- Single-node instances:

  Single-node instances do not support the backup function. You can use redis-cli to export data to RDB files. This operation depends on **SYNC** command.

  – If the instance allows the **SYNC** command, 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.

7.9 Why Is Memory of a DCS Redis Instance Unchanged After Data Migration Using Rump, Even If No Error Message Is Returned?

For details on how to use Rump, see the **Data Migration Guide**.

Possible causes:

- Rump does not support migration to cluster DCS instances.
- Commands are incorrectly run in Rump.

7.10 Where Are DCS Instance Backup Files Stored? How Many of Them Can Be Stored?

Backup files of DCS instances are stored to OBS. Currently, each DCS instance stores up to 24 latest backup files. The earliest ones will be automatically deleted if the amount exceeds.

7.11 Is All Data in a DCS Redis Instance Migrated During Online Migration?

Migration among multi-DB single-node, read/write splitting, or 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 by default. During the migration, data in all slots of DB0 is migrated.

7.12 When Will AOF Rewrites Be Triggered?

AOF rewrites involve the following concepts:

- Rewrite window, which is currently 01:00 to 04:59
- Disk usage threshold, which is 50%
- Dataset memory, which is the percentage of memory that Redis dataset has used.

AOF rewrites are triggered in the following scenarios:

- If the disk usage reaches the threshold (regardless of whether the current time is within the rewrite window), rewrites will be triggered on instances whose AOF file size is larger than the dataset memory.

- If the disk usage is below the threshold and the current time is within the rewrite window, rewrites will be triggered on instances whose AOF file size is larger than the dataset memory multiplied by 1.5.
- If the disk usage is below the threshold but the current time is out of the rewrite window, rewrites will be triggered on instances whose AOF file size is larger than the instance's maximum memory multiplied by 4.5.

7.13 What Are the Common Causes of Redis Migration Failures?

- Check if a master/standby switchover occurred during the migration. If it occurred, contact technical support to temporarily disable master/standby switchover until the migration completes.
- For online migration, check whether the **SYNC** and **PSYNC** commands are disabled on the source Redis instance. If they are disabled, enable them to allow data synchronization.
- By default, a Proxy Cluster instance has only one database (DB0). Before you migrate data from a single-node or master/standby instance to a Proxy Cluster instance, check whether any data exists on databases other than DB0. If yes, enable multi-DB for the Proxy Cluster instance by referring to **Enabling Multi-DB**.
- By default, a Redis Cluster instance has only one DB (DB0). Before you migrate data from a single-node or master/standby instance to a Redis Cluster instance, check whether any data exists on databases other than DB0. To ensure that the migration succeeds, move all data to DB0 by referring to **Online Migration with Rump**.

7.14 Can I Migrate Data to Multiple Target Instances in One Migration Task?

No. A migration task allows data to be migrated to only one target instance. To migrate data to multiple target instances, create multiple migration tasks.

7.15 How Do I Enable the SYNC and PSYNC Commands?

- Migration within DCS:
  - By default, the **SYNC** and **PSYNC** commands can be used when self-hosted Redis is migrated to DCS.
  - During online migration between DCS Redis instances in the same region under the same account, the **SYNC** and **PSYNC** commands are automatically enabled.
  - During online migration between DCS Redis instances in different regions or under different accounts within a region, the **SYNC** and **PSYNC** commands are not automatically enabled, and online migration cannot be used. You can migrate data using backup files.

- Migration from other cloud vendors to DCS:
    - Generally, cloud vendors disable the **SYNC** and **PSYNC** commands. If you want to use the online migration function on the DCS console, contact the O&M personnel of the source cloud vendor to enable the commands. For offline migration, you can import backup files.
    - If incremental migration is not required, you can perform full migration by referring to **Online Full Migration of Redis from Another Cloud with redis-shake**. This method does not depend on **SYNC** and **PSYNC**.

7.16 Will the Same Keys Be Overwritten During Data Migration or Backup Import?

If the data exists on both the source and target instances, the target data is overwritten by the source data. If the data exists only on the target instance, the data will be retained.

Inconsistency between source and target data after the migration may be due to the target data that existed and was retained before migration.

7.17 Why Does Redis Cluster Migration Fail If It Uses Built-in Keys and Cross-Slot Lua Scripts?

If your source Redis Cluster uses a cross-slot Lua script with built-in keys and it fails to be migrated to a cluster DCS instance, you can use a master/standby or read/write splitting instance as the target.

In scenarios where slot distribution changes, such as cluster scaling and slot migration, errors may incur in running a cross-slot Lua script with built-in keys. Therefore, **cross-slot Lua scripts with built-in keys are not recommended for Redis Cluster instances**.

☐ NOTE

Redis Cluster instances support cross-slot Lua scripts with built-in keys:
- Built-in keys: Keys are written in the Lua script instead of being input through a function.
- Cross-slot: All slots in a Lua script are on one shard.

Symptom

Online migration or backup import may fail when the source instance is a Redis Cluster that uses a cross-slot Lua script with built-in keys.

Solution

Select a master/standby or read/write splitting instance as the target.

Suggestion

Do not use cross-slot Lua scripts with built-in keys for Redis Cluster instances.

📖 NOTE

- Redis Cluster instances support cross-slot Lua scripts with built-in keys:
  - Built-in keys: Keys are written in the Lua script instead of being input through a function.
  - Cross-slot: All slots in a Lua script are on one shard.
- In scenarios where slot distribution changes, such as cluster scaling and slot migration, errors may incur in running a cross-slot Lua script with built-in keys.

7.18 Handling Migration Errors

This section provides solutions to common migration errors.

Restart data sync failed.

Solution:

1. **Check whether the source Redis has big keys**. If it does, split the big keys into small keys before migration.

2. Check the specifications of the target Redis instance and whether other tasks are being performed on the instance.
   - If the memory of the target Redis instance is smaller than the size of the data to be migrated, the memory will be used up during the migration and the migration will fail.
   - If a master/standby switchover is being performed on the target Redis instance, contact technical support to stop the master/standby switchover task and start it only after the data migration is completed.

3. Send the error information to technical support.

The Redis service address is unreachable.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Redis authentication failed.

Solution:

Ensure that the passwords of the source and target Redis databases are correct and are not changed during the migration.

If you forget the password, reconfigure the migration task after **resetting the password**.

RDB parsing failed.

Analysis:

Check the source Redis logs. Generally, this error is resulted from full output buffer when full synchronization takes too long or the size of incremental data is too large. You can use the following methods to solve this error:

- Increase the output buffer limit by **modifying the output-buffer-limit parameter**. This method is recommended.
- Increase the concurrency of redis-shake full synchronization by modifying the **parallel** parameter.
- Synchronize data during off-peak hours.

Failed to resume from the break point.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

IP address and port number of Redis are invalid.

Send the error information to technical support.

Job failed.

Send the error information to technical support.

Failed to download the file.

Solution:

See **Why Am I Unable to Download an Object?**

The cluster does not support import of AOF files.

Analysis:

Redis Cluster instances only support .rdb files.

Failed to migrate the AOF file to the target Redis.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**

- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Failed to migrate the RDB file to the target Redis.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Failed to decompress the file.

Solution:

1. Ensure that the file is not damaged and the file format is correct.
2. Check whether the specifications of the migration ECS are too small and the disk space is full. In this case, expand the specifications of the migration ECS.

The file format is not supported.

Analysis:

Only .rdb, .aof, .zip, and .tar.gz files are supported.

Failed to migrate files.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

No such file or directory.

Solution:

1. Check whether the specifications of the migration ECS are too small and the disk space is full. In this case, expand the specifications of the migration ECS.
2. Send the error information to technical support.

Failed to connect to the source Redis.

Solution:

1. See **Troubleshooting Redis Connection Failures**.
2. Check the specifications of the source Redis and the memory size of the migration ECS. If the memory of the migration server is small and the data volume of the source Redis is large, migration will be slow and data will be stacked on the migration ECS. To solve this error, you can expand the specifications of the migration ECS.
3. Run the **route - n** command on the migration ECS to check whether its route is normal.
4. Send the error information to technical support.

Failed to export the backup file from the source Redis.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Failed to import the backup file to the target Redis.

Check items:

- **Connection Between the Redis Instance and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Failed to modify the redis-shake-conf configuration file due to incorrect parameters.

Solution:

1. Check whether the specifications of the migration ECS are too small and the disk space is full. In this case, expand the specifications of the migration ECS.
2. Send the error information to technical support.

Data synchronization failed. Source node: {0}; target node: {1}.

Solution:

1. **Check whether the source Redis has big keys**. If it does, split the big keys into small keys before migration.

2. Ensure that the specifications of the target Redis instance are not smaller than those of the source Redis. For details about how to view specifications, see **Viewing Instance Details**.

3. See **Troubleshooting Redis Connection Failures**.

4. Check the specifications of the source Redis and the memory size of the migration ECS. If the memory of the migration server is small and the data volume of the source Redis is large, migration will be slow and data will be stacked on the migration ECS. To solve this error, you can expand the specifications of the migration ECS.

Failed to deploy the migration tool.

Solution:

1. Check whether the network between the data plane and OBS is normal.

2. Send the error information to technical support.

Online migration failed.

Send the error information to technical support.

Failed to bind the port to the ECS.

Solution:

The underlying resources are insufficient to support the migration task. Contact technical support.

Failed to create the migration ECS.

Send the error information to technical support.

File operation exception.

Solution:

1. Check the specifications of the source Redis and the memory size of the migration ECS. If the memory of the migration server is small and the data volume of the source Redis is large, migration will be slow and data will be stacked on the migration ECS. To solve this error, you can expand the specifications of the migration ECS.

2. Send the error information to technical support.

Command execution exception.

Solution:

● If the error information contains "listening-port" or "REPLCONF", check whether the **SYNC** and **PSYNC** commands are enabled on the source Redis instance and whether the underlying resources of the migration task are connected to the source and target Redis instances.

For online migration, the source and target Redis instances must be connected, and the **SYNC** and **PSYNC** commands must be enabled on the source Redis instance. Otherwise, the migration will fail.

– Check the network.

Check whether the source Redis instance, the target Redis instance, and the migration VM are configured with the same VPC. If they are in the same VPC, check the security group rules (for DCS Redis 3.0 instances) or whitelists (for DCS Redis 4.0 or 5.0 instances) to ensure that the IP addresses and ports of the Redis instances are accessible. If they are in different VPCs, **create a VPC peering connection**.

The source and target Redis instances must be accessible to the underlying VMs used for the migration task. For details about how to configure a whitelist, see **Managing IP Address Whitelist**.

If the source and target Redis instances are on different clouds, create a connection by referring to **Direct Connect documentation**.

– Check the commands.

By default, the **SYNC** and **PSYNC** commands are disabled by cloud vendors. To enable the commands, contact the O&M personnel of the cloud vendors.

▪ Migration within Huawei Cloud:

○ By default, the **SYNC** and **PSYNC** commands can be used when self-hosted Redis is migrated to DCS.

○ During online migration between Huawei Cloud DCS instances in the same region under the same account, the **SYNC** and **PSYNC** commands are automatically enabled.

○ During online migration between Huawei Cloud DCS instances in different regions or under different accounts within a region, the **SYNC** and **PSYNC** commands are not automatically enabled, and online migration cannot be used. You can migrate data using backup files instead.

▪ Migration from other cloud vendors to Huawei Cloud:

Generally, cloud vendors disable the **SYNC** and **PSYNC** commands. If you want to use online migration, contact the O&M personnel of the source cloud vendor to enable the commands. For offline migration, you can import backup files.

● If the error information contains "read error" and the migration failed during full migration due to the large data size, disable auto-reconnect before starting the migration and enable it after incremental migration starts. In addition, **increase the value of repl-timeout** and modify the output buffer of the source Redis based on the source memory size. For example, if the memory size of the source Redis is 24 GB, you can run the **client-output-buffer-limit slave 2gb 2gb 600** command to change the buffer size to 2 GB.

● If the error information contains "write: connection reset by peer", the target Redis memory may be too small. As a result, data cannot be synchronized when the target memory is full. In this case, set this parameter to **increase the specifications of the target Redis** to at least the same as the specifications of the source.

- If the error information contains "read: connection reset by peer", the source Redis is deployed in master/standby mode, and master/standby switchover occurs frequently during the migration. In this case, **check whether the source Redis has big keys**. If it does, split the big keys into small keys before migration. You can also run the **config set slave-priority 0** command to forcibly disable master/standby switchover and enable it after the migration is complete. If the target Redis instance is Proxy Cluster, check the size of the pipeline. Run the **proxy.config set client-max-pipeline 50000** command to change this limit to 50,000 for proxies.

- Send the error information to technical support.

Decoding or parsing failed.

Solution:

1. Check whether the specifications of the migration ECS are too small and the disk space is full. In this case, expand the specifications of the migration ECS.

2. Send the error information to technical support.

Unknown or unsupported command.

Solution:

Check whether the **SYNC** and **PSYNC** commands are enabled on the source Redis instance. If they are not enabled, contact technical support.

For online migration, the source and target Redis instances must be connected, and the **SYNC** and **PSYNC** commands must be enabled on the source Redis instance. Otherwise, the migration will fail.

- Check the network.

  Check whether the source Redis instance, the target Redis instance, and the migration VM are configured with the same VPC. If they are in the same VPC, check the security group rules (for DCS Redis 3.0 instances) or whitelists (for DCS Redis 4.0 or 5.0 instances) to ensure that the IP addresses and ports of the Redis instances are accessible. If they are in different VPCs, **create a VPC peering connection**.

  The source and target Redis instances must be accessible to the underlying VMs used for the migration task. For details about how to configure a whitelist, see **Managing IP Address Whitelist**.

  If the source and target Redis instances are on different clouds, create a connection by referring to **Direct Connect documentation**.

- Check the commands.

  By default, the **SYNC** and **PSYNC** commands are disabled by cloud vendors. To enable the commands, contact the O&M personnel of the cloud vendors.

  - Migration within Huawei Cloud:

    - By default, the **SYNC** and **PSYNC** commands can be used when self-hosted Redis is migrated to DCS.

    - During online migration between Huawei Cloud DCS instances in the same region under the same account, the **SYNC** and **PSYNC** commands are automatically enabled.

- During online migration between Huawei Cloud DCS instances in different regions or under different accounts within a region, the **SYNC** and **PSYNC** commands are not automatically enabled, and online migration cannot be used. You can migrate data using backup files instead.

  - Migration from other cloud vendors to Huawei Cloud:

    Generally, cloud vendors disable the **SYNC** and **PSYNC** commands. If you want to use online migration, contact the O&M personnel of the source cloud vendor to enable the commands. For offline migration, you can import backup files.

Data synchronization failed.

Solution:

1. If the error information contains "key name is busy", a key with the same name already exists in the target Redis. In this case, delete the key.

2. If the error information contains "not in the same slot", reconstruct your service. Do not use cross-slot keys in a multi-key command. You can also use a master/standby instance instead of a Proxy Cluster instance as the target.

3. If the error information contains "read: connection reset by peer", the source Redis is deployed in master/standby mode, and master/standby switchover occurs frequently during the migration. In this case, **check whether the source Redis has big keys**. If it does, split the big keys into small keys before migration. You can also run the **config set slave-priority 0** command to forcibly disable master/standby switchover and enable it after the migration is complete. If the target Redis instance is Proxy Cluster, check the size of the pipeline. Run the **proxy.config set client-max-pipeline 50000** command to change this limit to 50,000 for proxies.

Failed to import the backup file.

Send the error information to technical support.
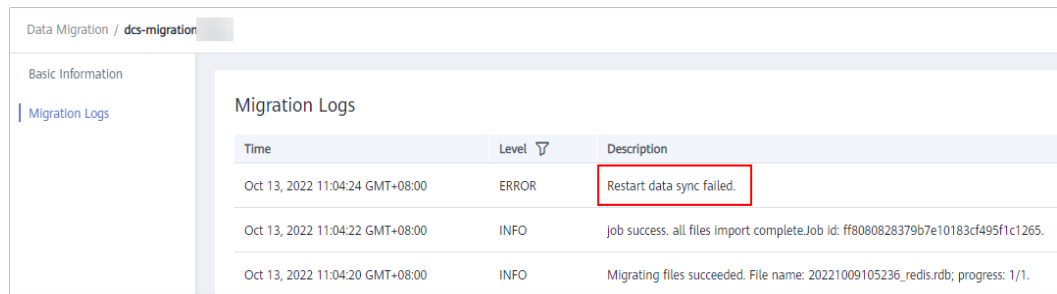
7.19 Troubleshooting Data Migration Failures

When you use the console to migrate data, the migration may fail if you select an inappropriate migration scheme, the **SYNC** and **PSYNC** commands are not allowed on the source Redis instance, or the network between the source and target Redis instances is disconnected.

This section describes how to troubleshoot data migration failures on the DCS console.

Procedure

Step 1 Click the name of a migration task and then go to the **Migration Logs** page.

Step 2 Check the migration logs. Rectify the fault based on the error log. For details, see **Handling Migration Errors**.

Figure 7-1 Viewing migration logs

Data Migration / dcs-migration			
Basic Information	Migration Logs		
Migration Logs	Time	Level	Description
	Oct 13, 2022 11:04:24 GMT+08:00	ERROR	Restart data sync failed.
	Oct 13, 2022 11:04:22 GMT+08:00	INFO	job success. all files import complete.Job id: ff8080828379b7e10183cf495f1c1265.
	Oct 13, 2022 11:04:20 GMT+08:00	INFO	Migrating files succeeded. File name: 20221009105236_redis.rdb; progress: 1/1.

Step 3 Check whether you used the appropriate migration scheme.

An appropriate scheme varies depending on the scenario. That is, the scheme for migrating data from self-hosted Redis to a DCS instance, from other vendors' Redis to a DCS instance, and between DCS instances are different. If you migrate data between DCS instances, the source instance cannot be a higher version than the target instance.

If the migration scheme is inappropriate, the migration will fail. For details about migration schemes, see **Migration Tools and Schemes**.

Step 4 Check whether the **SYNC** and **PSYNC** commands are enabled on the source Redis instance and whether the underlying resources of the migration task are connected to the source and target Redis instances.

This operation is required only for online migration.

For online migration, the source and target Redis instances must be connected, and the **SYNC** and **PSYNC** commands must be enabled on the source Redis instance. Otherwise, the migration will fail.

- Check the network.

  Check whether the source Redis instance, the target Redis instance, and the migration VM are configured with the same VPC. If they are in the same VPC, check the whitelists to ensure that the IP addresses and ports of the Redis instances are accessible. If they are in different VPCs, **create a VPC peering connection**.

  The source and target Redis instances must be accessible to the underlying VMs used for the migration task. For details about how to configure a whitelist, see **Managing IP Address Whitelist**.

  To allow the VM resources used by the migration task to access the source and target Redis instances, outbound rules of the migration security group must allow traffic over the IP addresses and ports of the source and target instances. For details, see **How Do I Configure a Security Group?**

  If the source and target Redis instances are on different clouds, create a connection by referring to **Direct Connect documentation**.

  ☐ **NOTE**

  The VM used by a migration task uses an IP address. Configuring the whitelist (Redis 4.0/5.0/6.0 basic) of instances is to allow the migration VM to access the source and target Redis.

- Check the commands.

By default, the **SYNC** and **PSYNC** commands are disabled by cloud vendors. To enable the commands, contact the O&M personnel of the cloud vendors.

- – Migration within HUAWEI CLOUD:

  - By default, the **SYNC** and **PSYNC** commands can be used when self-hosted Redis is migrated to DCS.

  - During online migration between Huawei Cloud DCS instances in the same region under the same account, the **SYNC** and **PSYNC** commands are automatically enabled.

  - During online migration between Huawei Cloud DCS instances in different regions or under different accounts within a region, the **SYNC** and **PSYNC** commands are not automatically enabled, and online migration cannot be used. You can migrate data using backup files instead.

- – Migration from other cloud vendors to HUAWEI CLOUD:

  Generally, cloud vendors disable the **SYNC** and **PSYNC** commands. If you want to use online migration, contact the O&M personnel of the source cloud vendor to enable the commands. For offline migration, you can import backup files.

Step 5 Check the source Redis instance for big keys. For details, see **Analyzing Big Keys and Hot Keys**.

If the source Redis instance has big keys, split them into small keys before migration.

Step 6 Check the specifications of the target Redis instance and whether other tasks are being performed on the instance.

If the memory of the target Redis instance is smaller than the size of the data to be migrated, the memory will be used up during the migration and the migration will fail.

If a master/standby switchover is being performed on the target Redis instance, contact technical support to stop the master/standby switchover task and start it only after the data migration is completed.

Step 7 If you migrate data from a single-node or master/standby instance to a cluster instance, check the following items:

- By default, a Proxy Cluster instance has only one database (DB0). Before you migrate data from a single-node or master/standby instance to a Proxy Cluster instance, check whether any data exists on databases other than DB0. If yes, enable multi-DB for the Proxy Cluster instance by referring to **Enabling Multi-DB**.

- By default, a Redis Cluster instance has only one DB (DB0). Before you migrate data from a single-node or master/standby instance to a Redis Cluster instance, check whether any data exists on databases other than DB0. To ensure that the migration succeeds, move all data to DB0 by referring to **Online Migration with Rump**.

Step 8 Check whether the migration task is performed correctly.

Check whether the IP address and the instance password are correct.

Step 9 Check the whitelist.

Step 10 If the fault persists, contact technical support.

----End

8 Big/Hot Key Analysis

8.1 What Are Big Keys and Hot Keys?

Term	Definition
Big key	There are two types of big keys: ● Keys that have a large value. If the size of a single String key exceeds 10 KB, or if the size of all elements of a key combined exceeds 50 MB, the key is defined as a big key. ● Keys that have a large number of elements. If the number of elements in a key exceeds 5000, the key is defined as a big key.
Hot key	A key is defined as a hot key if it is frequently requested or if it occupies a large number of resources. For example: ● In a cluster instance, a shard processes 10,000 requests per second, among which 3000 are performed on the same key. ● In a cluster instance, a shard uses a total of 100 Mbits/s inbound and outbound bandwidth, among which 80 Mbits/s is used by the **HGETALL** operation on a Hash key.

8.2 What Is the Impact of Big Keys or Hot Keys?

Category	Impact
Big key	**Instance specifications fail to be modified.** Specification modification of a Redis Cluster instance involves rebalancing (data migration between nodes). Redis has a limit on key migration. If the instance has any single key bigger than 512 MB, the modification will fail when big key migration between nodes times out. The bigger the key, the more likely the migration will fail.

Category	Impact
	Data migration fails. During data migration, if a key has many elements, other keys will be blocked and will be stored in the memory buffer of the migration ECS. If they are blocked for a long time, the migration will fail.
	Cluster shards are unbalanced. ● The memory usage of shards is unbalanced. For example, if a shard uses a large memory or even uses up the memory, keys on this shard are evicted, and resources of other shards are wasted. ● The bandwidth usage of shards is unbalanced. For example, flow control is repeatedly triggered on a shard.
	Latency of client command execution increases. Slow operations on a big key block other commands, resulting in a large number of slow queries.
	Flow control is triggered on the instance. Frequently reading data from big keys exhausts the outbound bandwidth of the instance, triggering flow control. As a result, a large number of commands time out or slow queries occur, affecting services.
	Master/standby switchover is triggered. If the high-risk **DEL** operation is performed on a big key, the master node may be blocked for a long time, causing a master/standby switchover.
Hot key	**Cluster shards are unbalanced.** If only the shard where the hot key is located is busy processing service queries, there may be performance bottlenecks on a single shard, and the compute resources of other shards may be wasted.
	CPU usage surges. A large number of operations on hot keys may cause high CPU usage. If the operations are on a single cluster shard, the CPU usage of the shard where the hot key is located will surge. This will slow down other requests and the overall performance. If the service volume increases sharply, a master/standby switchover may be triggered.
	Cache breakdown may occur. If Redis cannot handle the pressure on hot keys, requests will hit the database. The database may break down as its load increases dramatically, affecting other services.

8.3 How Do I Avoid Big Keys and Hot Keys?

- **Keep the size of Strings within 10 KB** and **the quantity** of Hashes, Lists, Sets, or Zsets **within 5000**.

- When naming keys, use the service name abbreviation as the prefix and do not use special characters such as spaces, line brakes, single or double quotation marks, and other escape characters.

- Do not rely too much on Redis transactions.

- The performance of short connections is poor. Use clients with connection pools.

- Do not enable data persistence if you use Redis just for caching and can tolerate data losses.

- For details about how to optimize big keys and hot keys, see the following table.

Category	Method
Big key	**Split big keys.** Scenarios: - **If the big key is a String**, you can split it into several key-value pairs and use **MGET** or a pipeline consisting of multiple **GET** operations to obtain the values. In this way, the pressure of a single operation can be split. For a cluster instance, the operation pressure can be evenly distributed to multiple shards, reducing the impact on a single shard. - **If the big key contains multiple elements, and the elements must be operated together**, the big key cannot be split. You can remove the big key from Redis and store it on other storage media instead. This scenario should be avoided by design. - **If the big key contains multiple elements, and only some elements need to be operated each time**, separate the elements. Take a Hash key as an example. Each time you run the **HGET** or **HSET** command, the result of the hash value modulo N (customized on the client) determines which key the field falls on. This algorithm is similar to that used for calculating slots in Redis Cluster.
	Store big keys on other storage media. If a big key cannot be split, it is not suitable to be stored in Redis. You can store it on other storage media, such as NoSQL databases, and delete the big key from Redis. **CAUTION** Do not use the **DEL** command to delete big keys. Otherwise, Redis may be blocked or even a master/standby switchover may occur.

Category	Method
	Set appropriate expiration and delete expired data regularly. Appropriate expiration prevents expired data from remaining in Redis. Due to Redis's lazy free, expired data may not be deleted in time. If this occurs, **scan expired keys**.
Hot key	**Split read and write requests.** If a hot key is frequently read, **configure read/write splitting** on the client to reduce the impact on the master node. You can also add replicas to meet the read requirements, but there cannot be too many replicas. In DCS, replicas replicate data from the master in parallel. The replicas are independent of each other and the replication delay is short. However, if there is a large number of replicas, CPU usage and network load on the master node will be high.
	Use the client cache or local cache. If you know what keys are frequently used, you can design a two-level cache architecture (client/local cache and remote Redis). Frequently used data is obtained from the local cache first. The local cache and remote cache are updated with data writes at the same time. In this way, the read pressure on frequently accessed data can be separated. This method is costly because it requires changes to the client architecture and code.
	Design a circuit breaker or degradation mechanism. Hot keys can easily result in cache breakdown. During peak hours, requests are passed through to the backend database, causing service avalanche. To ensure availability, the system must have a circuit breaker or degradation mechanism to limit the traffic and degrade services if breakdown occurs.

8.4 How Do I Detect Big Keys and Hot Keys in Advance?

Method	Description
Through **Big Key Analysis** and **Hot Key Analysis** on the DCS console	See **Analyzing Big Keys and Hot Keys**.

Method	Description
By using the **bigkeys** and **hotkeys** options on redis-cli	• redis-cli uses the **bigkeys** option to traverse all keys in a Redis instance and returns the overall key statistics and the biggest key of six data types: Strings, Lists, Hashes, Sets, Zsets, and Streams. The command is **redis-cli -h** *<Instance connection address>* **-p** *<Port number>* **-a** *<Password>* **--bigkeys**. • In Redis 4.0 and later, you can use the **hotkeys** option to quickly find hot keys in redis-cli. Run this command during service running to find hot keys: **redis-cli -h** *<Instance connection address>* **-p** *<Port number>* **-a** *<Password>* **--hotkeys**. The hot key details can be obtained from the summary part in the returned result.
Searching for big keys using Redis commands	If there is a pattern of big keys, for example, the prefix is **cloud:msg:test**, you can use a program to scan for keys that match the prefix, and then run commands to query the number of members in the key and query the key sizes to find big keys. • Commands for querying the number of members: **LLEN**, **HLEN**, **XLEN**, **ZCARD**, **SCARD** • Commands for querying the memory usage of keys: **DEBUG OBJECT**, **MEMORY USAGE** **CAUTION** This method consumes a large number of computing resources. To ensure service running, do not use this method on instances with heavy service pressure.
Searching for big keys using redis-rdb-tools	**redis-rdb-tools** is an open-source tool for analyzing Redis RDB snapshot files. You can use it to analyze the memory usage of all keys in a Redis instance. To use this method, you must **export the RDB file of an instance** on the **Backups & Restorations** page of the DCS console. **CAUTION** This method does not affect service running, but is not as timely as online analysis.

8.5 How Does DCS Delete Expired Keys?

Question

What are the rules for scheduled deletion of expired keys on a daily basis? Can I customize the rules?

Mechanisms for Deleting Expired Keys

- Lazy free deletion: The deletion strategy is controlled in the main I/O event loop. Before a read/write command is executed, a function is called to check whether the key to be accessed has expired. If it has expired, it will be deleted and a response will be returned indicating that the key does not exist. If the key has not expired, the command execution resumes.

- Scheduled deletion: A time event function is executed at certain intervals. Each time the function is executed, a random collection of keys are checked, and expired keys are deleted.

  📖 **NOTE**

  To avoid prolonged blocks on the Redis main thread, not all keys are checked in each time event. Instead, a random collection of keys are checked each time. As a result, the memory used by expired keys cannot be released quickly.

Solutions

- Configure scheduled hot key analysis tasks by referring to **Hot Key Analysis**, or use the **SCAN** command to traverse all keys on a scheduled basis and remove expired keys from the memory.

- Configure a scheduled task to scan all master nodes of the instance. All keys will be scanned, and Redis will determine whether the keys have expired. Expired keys will be released. For details, see **Scanning Expired Keys**.

How Do I Know Which Expired Keys Have Been Deleted?

Deleted expired keys cannot be queried.

8.6 How Long Are Keys Stored? How Do I Set Key Expiration?

- Key storage duration
  - Keys that do not have an expiration are stored permanently.
  - Keys that have an expiration are deleted after they expire. For details, see **Scanning Expired Keys**.
  - To remove the expiration set for a key, run the **PERSIST** command.

- Setting key expiration

  You can run the **EXPIRE** or **PEXPIRE** command to set the key expiration time. For example, if you run **expire key1 100**, key1 will expire in 100 seconds. If you run **pexpire key2 1800**, key2 will expire in 1800 milliseconds.

  EXPIRE sets key expiration in seconds, and **PEXPIRE** sets key expiration in milliseconds.

8.7 Why Does Memory Usage Decrease After Big Key Analysis Is Performed on Redis?

Big key analysis only queries keys that occupy a large space and does not delete the keys. Due to the lazy free deletion mechanism, keys are not deleted

immediately upon expiration, unless they are accessed or identified. During a big key analysis, all keys are traversed, and the expired keys will be identified and then deleted. As a result, memory usage decreases after a big key analysis is completed.

For details about how expired keys are deleted and how to scan for expired keys, see **Scanning Expired Keys**.

9 Redis Commands

9.1 Does DCS for Redis Support Command Audits?

No. To ensure high-performance read and write operations, DCS for Redis does not audit commands. Commands are not printed.

9.2 How Do I Clear Redis Data?

Exercise caution when clearing data.

- Redis 4.0 or later

  To clear data of a DCS Redis 4.0 or later instance, you can run the **FLUSHDB** or **FLUSHALL** command in redis-cli, use the data clearing function on the DCS console, or run the **FLUSHDB** command on Web CLI.

  To clear data of a Redis Cluster instance, run the **FLUSHDB** or **FLUSHALL** command on every shard of the instance. Otherwise, data may not be completely cleared.

  > 📖 **NOTE**
  >
  > - When you run the **FLUSHDB** command on Web CLI, only one shard is cleared at a time. If there are multiple shards, connect to the master node of each shard and run the **FLUSHDB** command separately.
  > - Redis Cluster data cannot be cleared by using Web CLI.

9.3 How Do I Find Specified Keys and Traverse All Keys?

Finding Specified Keys

Big key and hot key analysis does not support key searching with specified conditions. To find keys with the specified prefix or suffix, use the **SCAN** command.

For example, to search for keys that contain the letter *a* in a Redis instance, run the following command in redis-cli:

./redis-cli -h *{redis_address}* **-p** *{port}* **[-a** *password***] --scan --pattern '*a*'**

Traversing All Keys

Do not use the **KEYS** command to traverse all keys of an instance because the **KEYS** command is complex and may block Redis. To traverse all keys in a DCS Redis instance, run the following command in redis-cli:

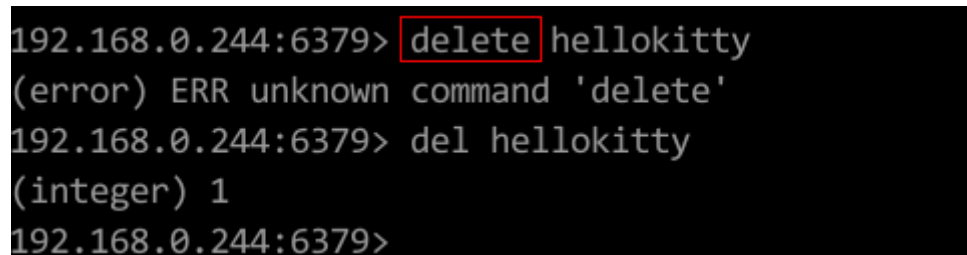**./redis-cli -h** *{redis_address}* **-p** *{port}* **[-a** *password***] --scan --pattern '*'**

For details about the **SCAN** command, visit the **Redis official website**.

9.4 Why Do I Fail to Execute Some Redis Commands?

Possible causes include the following:

- The command is spelled incorrectly.

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

  ```
  192.168.0.244:6379> delete hellokitty
  (error) ERR unknown command 'delete'
  192.168.0.244:6379> del hellokitty
  (integer) 1
  192.168.0.244:6379>
  ```

- A command available in a higher Redis version is run in a lower Redis version.
- 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 command cannot be executed in Web CLI.

  In addition to the disabled and restricted Redis commands, the **KEYS** commands is also restricted in Web CLI.

- 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, submit a service ticket for the instance to be upgraded.

- The **CLIENT SETNAME** and **CLIENT GETNAME** commands fail to be executed.

  The DCS Redis instance is of a lower version that does not support these commands. In this case, submit a service ticket for the instance to be upgraded.

9.5 Why is "permission denied" Returned When I Run the Keys Command in Web CLI?

The **KEYS** command is disabled in Web CLI. This command can only be run in redis-cli.

9.6 How Do I Rename High-Risk Commands?

Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, **HGETALL**, **SCAN**, **HSCAN**, **SSCAN**, and **ZSCAN** commands. For details, see **Renaming Commands**.

☐ NOTE

- Currently, Redis does not support disabling of commands. To avoid risks when using the preceding commands, rename them. For details about the supported and disabled commands in DCS, see **Command Compatibility**.

- After you confirm the modification, the system will restart the instance. The new commands take effect after the restart. Remember the new command names because they will not be displayed on the console for security purposes.

9.7 Does DCS for Redis Support Pipelining?

Yes.

For Redis Cluster instances, ensure that all commands in a pipeline are executed on the same shard.

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

Yes.

For more information about Redis command compatibility, see **Redis Command Compatibility**.

9.9 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.

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

Redis command timeouts can be controlled on the client end or server end.

- Timeouts on the client end are controlled in the client code. You can determine the timeouts that suit service needs. For example, if you use Lettuce, a Java client, configure the **timeout** parameter.

- On the server end, the **timeout** parameter is set to **0** by default, indicating that connections will never be terminated. Modify the parameter setting by referring to **Modifying Configuration Parameters**.

9.11 Can I Configure Redis Keys to Be Case-Insensitive?

No. Like in open-source Redis, keys in DCS for Redis are case-sensitive and cannot be configured to be case-insensitive.

9.12 Common Web CLI Errors

1. ERR Wrong number of arguments for 'xxx' command

   This error indicates that the executed Redis command has a parameter error (syntax error). Rewrite the command by referring to the open-source Redis command protocol.

2. ERR unknown command 'xxx'

   This error indicates that the command is unknown or is not a valid command defined by Redis. Rewrite the command by referring to the open-source Redis command protocol.

3. ERR Unsupported command: 'xxx'

   This error indicates that the command is disabled for DCS Redis instances. For details, see **Web CLI Commands**.

10 Monitoring and Alarm

10.1 Why Is CPU Usage of a DCS Redis Instance 100%?

Symptom

The CPU usage of a Redis instance increases dramatically within a short period of time. If the CPU usage is too high, connections may time out, and master/standby switchover may be triggered.

Possible Causes

1. The service QPS is high. In this case, refer to **Checking QPS**.
2. Resource-consuming commands, such as **KEYS**, were used. In this case, refer to **Locating and Disabling CPU-Intensive Commands**.
3. Redis rewrite was triggered. In this case, refer to **Checking Redis Rewrite**.

Checking QPS

On the **Cache Manager** page of the DCS console, click an instance to go to the instance details page. On the left menu, choose **Performance Monitoring** and then view the **Ops per Second** metric.

If the QPS is high, optimize customer services or **modifying instance specifications**. For details about the QPS supported by different instance specifications, see **DCS Instance Specifications**.

Locating and Disabling CPU-Intensive Commands

Resource-consuming commands (commands with time complexity O(N) or higher), such as **KEYS**, are used. Generally, the higher the time complexity, the more resources a command uses. As a result, the CPU usage is high, and a master/standby switchover can be easily triggered. For details about the time complexity of each command, visit the **Redis official website**. In this case, use the **SCAN** command instead or disable the **KEYS** command.

Step 1 On the **Performance Monitoring** page of the DCS console, locate the period when the CPU usage is high.

Step 2 Use the following methods to find the commands that consume a large number of resources.

- Redis logs queries that exceed a specified execution duration. You can find the commands that consume a large number of resources by analyzing the slow queries and their execution duration. For details, see **Viewing Redis Slow Queries**.

- Use the instance diagnosis function to analyze the execution duration percentage of different commands during the period when the CPU usage is high. For details, see **Diagnosing an Instance**.

Step 3 Resolve the problem.

- Evaluate and disable high-risk and high-consumption commands, such as **FLUSHALL**, **KEYS**, and **HGETALL**.

- Optimize services. For example, avoid frequent data sorting operations.

- (Optional) Perform the following operations to adjust instances based on service requirements:

  – Change the instance type to read/write splitting to separate read and write requests from high-consumption commands or applications.

  – Scale up the instance.

  ----End

Checking Redis Rewrite

AOF persistence, which is enabled by default for master/standby and cluster DCS Redis instances, takes place in the following scenarios:

- If a small amount of data is written and the AOF file is not large, AOF rewrite is performed from 01:00 to 04:00 in the morning every day, and CPU usage may suddenly spike during this period.

- When a large amount of data is written and the AOF file size exceeds the threshold (three to five times the DCS instance capacity), AOF rewrite is automatically triggered in the background regardless of the current time.

Redis rewrite is performed by running the **BGSAVE** or **BGREWRITEAOF** command, which may consume many CPU resources (see **the discussion**). **BGSAVE** and **BGREWRITEAOF** commands need to fork(), resulting in CPU usage spikes within a short period of time.

If persistence is not required, disable it by changing the value of **appendonly** to **no** on the **Parameters** page of the instance. However, if you disable persistence, data loss may occur due to a lack of data flushing to disk in extreme situations.

10.2 How Do I View Current Concurrent Connections and Maximum Connections of a DCS Redis Instance?

Viewing Concurrent Connections of a DCS Redis Instance

The number of real-time connections received by a DCS instance is a metric that can be viewed on the console. For details on how to view the metrics, see **Viewing DCS Monitoring Metrics**.

On the Cloud Eye console, find the **Connected Clients** metric. Click [icon] to view monitoring details on an enlarged graph.

Specify a time range to view the metric in a specific monitoring period. For example, you can select a 10-minute period to view the number of connections received during the period. On the graph, you can view the trend and the total number of connections received during the period.

📖 **NOTE**

The **Connected Clients** metric means the number of connected clients. It includes connections established for system monitoring, configuration synchronization, and services, but excludes connections from replicas.

Viewing or Modifying the Maximum Connections of an Instance

You can view the default and maximum allowed number of connections on the instance creation page or in the **document**.

After an instance is created, you can view or change the value of **maxclients** (the maximum number of connections) on the **Instance Configuration** > **Parameters** page of the instance. (Proxy Cluster instances do not have this parameter.)

If the limit is exceeded, excess requests will be rejected and connections will time out.

10.3 What Should I Do If the Monitoring Data of a DCS Redis Instance Is Abnormal?

If you have any doubt on the monitoring data of a DCS Redis instance, you can access the instance through redis-cli and run the **INFO ALL** command to view the metrics. For details about the output of the **INFO ALL** command, see **https://redis.io/docs/latest/commands/info/**.

10.4 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.

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.

10.5 Why Does Bandwidth Usage Exceed 100%?

The basic information about the bandwidth usage metric is as follows.

Metric ID	Metric Name	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Monitored object: Redis 4.0 and later Redis Server of a master/standby, read/write splitting, or cluster instance Dimension: dcs_cluster_node	1 minute

Bandwidth usage = (Input flow + Output flow)/(2 x Maximum bandwidth) x 100%

According to the formula, the bandwidth usage counts in the input flow and output flow, which include the traffic for replication between the master and replicas. Therefore, the total bandwidth usage is larger than the normal service traffic, and may exceed 100%.

If the value of the **Flow Control Times** metric is larger than 0, the maximum bandwidth has been reached and flow control has been performed.

However, flow control decisions are made without considering the traffic for replication between the master and replicas. Therefore, sometimes the bandwidth usage exceeds 100% but the number of flow control times is 0.

10.6 Why Is the Rejected Connections Metric Displayed?

If the **Rejected Connections** metric is displayed, check if the number of connected clients exceeds the maximum allowed number of connections of the instances.

- To check the maximum allowed number of connections, go to the **Parameters** tab page of the instance and check the value of the **maxclients** parameter.

- To check the current number of connections, go to the **Performance Monitoring** tab page of the instance and check the **Connected Clients** metric.

If the current number of connections reaches the upper limit, you can adjust the value of **maxclients**. If the value of **maxclients** can no longer be increased, increase the instance specifications.

10.7 Why Is Flow Control Triggered? How Do I Handle It?

Flow control is triggered when the traffic used by a Redis instance in a period exceeds the maximum bandwidth. Connections may be discarded due to flow control, resulting in high service latency and client connection exceptions.

📖 **NOTE**

For details about the maximum allowed bandwidth, see the "Assured/Maximum Bandwidth" column of different instance types listed in **DCS Instance Specifications**.

Even if the bandwidth usage is low, flow control may still be triggered. The real-time bandwidth usage is reported once in each reporting period. Flow controls are checked every second. The traffic may surge within seconds and then fall back between reporting periods. By the time the bandwidth usage is reported, it may have already restored to the normal level.

For master/standby instances:

- If flow control is always triggered when the bandwidth usage is low, there may be service microbursts or big or hot keys. In this case, check for big or hot keys.

- If the bandwidth usage remains high, the bandwidth limit may be exceeded. In this case, expand the capacity.

For cluster instances:

- If flow control is triggered only on one or a few shards, the shards may have big or hot keys.

- If flow control or high bandwidth usage occurs on all or most shards at the same time, bandwidth usage of the instance has reached the limit. In this case, expand the instance capacity.

□ NOTE

- Perform big key and hot key analysis on the DCS console, and take measures accordingly. For details, see **Analyzing Big Keys and Hot Keys**.

- Running commands (such as **KEYS**) that consume lots of resources may cause high CPU and bandwidth usage. As a result, flow control is triggered.

11 Master/Standby Switchover

11.1 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.
- A master/standby switchover will be triggered when the master node of a master/standby instance fails.

  For example, if commands (such as **KEYS**) that consume a lot of resources are used or logs are aged and deleted in batches, the CPU usage will surge, triggering a master/standby switchover.

- If you restart a master/standby instance on the DCS console, a master/standby switchover will be triggered.
- If you scale up a single-node, read/write splitting, or master/standby instance, a master/standby switchover will be triggered.

  During scale-up, a new standby node with the new specifications is created. After full and incremental data on the master node is synchronized to the standby node, a master/standby switchover is performed and the original node is deleted.

After a master/standby switchover occurs, you will receive a notification. Check whether the client services are running properly. If not, check whether the TCP connection is normal and whether it can be re-established after the master/standby switchover to restore the services.

11.2 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.

11.3 Does the Client Need to Switch the Connection Address After a Master/Standby Switchover?

No. If the master node is fails or a master/standby switchover is performed, the standby node will be promoted to master and takes the original IP address.

11.4 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.

12 Instance Creation and Permissions

12.1 Why Do I Fail to Create a DCS Redis Instance?

- The subnet does not have sufficient IP addresses.

  Analysis: Each node in a DCS instance must be assigned an IP address. Therefore, a single-node instance requires one IP address, a master/standby instance requires two IP addresses, and a cluster instance requires multiple IP addresses.

  Solution: Create the instance in a different subnet within the VPC or release IP addresses in the current subnet.

- The IAM user does not have the permissions required to create an instance.

  Analysis: The group to which the user belongs must be granted the **DCS FullAccess** policy or **DCS Administrator** role or other policies containing the permissions required for creating DCS instances.

  Solution: Create a DCS instance as the administrator.

12.2 Why Can't I View the Subnet and Security Group Information When Creating a DCS Instance?

This may be because you do not have the **Server Administrator** and **VPC Administrator** roles. For details on how to add user permissions, see **Modifying User Group Permissions**.

12.3 Why Can't I Select the Required Enterprise Project When Creating a DCS Instance?

Symptom

The desired enterprise project is not displayed during instance creation.

Cause

The user group does not have DCS permissions in the desired enterprise project.

Solution

1. Log in to the DCS console.

2. In the upper right corner, choose **Enterprise** > **Project Management**. On the displayed page, click **View Resource** in row containing the desired enterprise project.

3. Click the **Permissions** tab. Then, click **Authorize User Group**.

   📖 **NOTE**

   Click **Authorize User Group** to grant permissions to a user group, or click **Authorize User** to grant permissions to a user.

4. Click **Authorize** in the row containing the user or user group to which you want to grant permissions.

5. Search for and select the **DCS FullAccess** policy, click **Next**, and click **OK**.

   For more information about DCS permissions policies, see **Permissions Management**.

   📖 **NOTE**

   If you configure both the **DCS UserAccess** (containing deny statements) and **DCS FullAccess** policies, you cannot create, modify, delete, or scale DCS instances because deny statements will take precedence. To perform the operations allowed by **DCS FullAccess**, delete **DCS UserAccess** first.

12.4 Why Can't an IAM User See a New DCS Redis Instance?

Symptom

An IAM user cannot see a newly created DCS Redis instance.

Possible Cause

The IAM user does not have permissions for the enterprise project to which the new instance belongs.

Solution

1. Log in to the DCS console.

2. In the upper right corner, choose **Enterprise** > **Project Management**. On the displayed page, click **View Resource** in row containing the desired enterprise project.

3. Click the **Permissions** tab. Then, click **Assign Permissions** on the **User Groups** tab.

4. Select user groups you want to assign permissions to, and click **Next**.

5. Select **DCS UserAccess** and click **OK**.