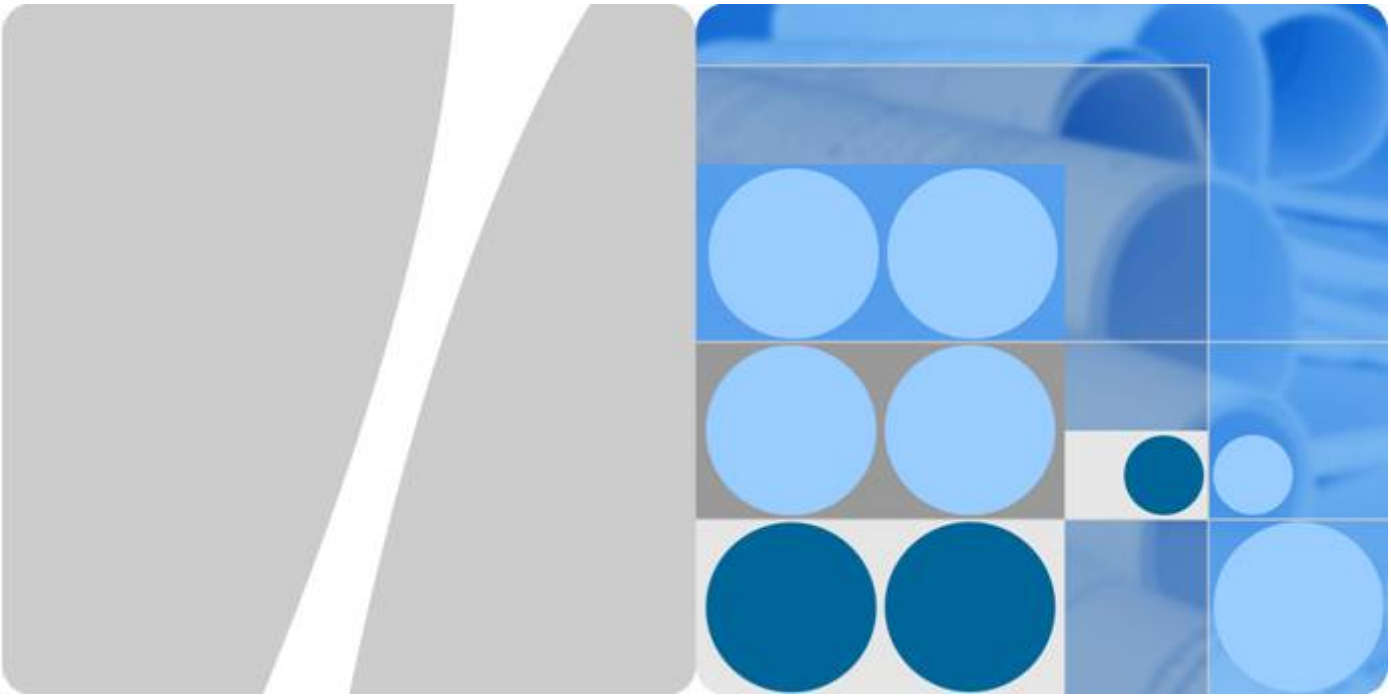


Distributed Cache Service

Best Practices

Issue 01

Date 2022-01-17

HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

Copyright © Huawei Technologies Co., Ltd. 2022. All rights reserved.

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

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Contents

1 Serializing Access to Frequently Accessed Resources..... 1
2 Ranking with Redis..... 7
3 Suggestions on Using Redis..... 10
4 Using Nginx for Public Access to Single-node, Master/Standby, or Proxy Cluster DCS Redis 4.0 or 5.0 Instances..... 13
5 Using SSH Tunneling for Public Access to a DCS Instance..... 18
6 Using ELB for Public Access to DCS Across VPCs..... 22

 Copyright © Huawei Technologies Co., Ltd.

1 Serializing Access to Frequently Accessed Resources

Background

In monolithic deployment, you can use Java concurrency APIs such as **ReentrantLock** or **synchronized** to implement mutual exclusion locks. This native lock mechanism provided by Java ensures that multiple threads within a Java VM process are executed concurrently and sequentially.

However, this mechanism may fail in multi-node deployment because a node's lock only takes effect on threads in the Java VM where the node runs. For example, the concurrency level in Internet seckills requires multiple nodes to run at the same time. Assume that requests of two users arrive simultaneously on two nodes. Although the requests can be processed simultaneously on the respective nodes, an inventory oversold problem may still occur because the nodes use different locks.

Solution

To serialize access to resources, ensure that all nodes use the same lock. This requires a distributed lock.

The idea of a distributed lock is to provide a globally unique "thing" for different systems to obtain locks. When a system needs a lock, it asks the "thing" for a lock. In this way, different systems can obtain the same lock.

Currently, a distributed lock can be implemented using cache databases, disk databases, or ZooKeeper.

Implementing distributed locks using DCS Redis instances has the following advantages:

- Simple operation: Locks can be acquired and released by using simple commands such as **SET**, **GET**, and **DEL**.
- High performance: Cache databases deliver higher read/write performance than disk databases and ZooKeeper.
- High reliability: DCS supports both master/standby and cluster instances, preventing single points of failure.

 Copyright © Huawei Technologies Co., Ltd.

Implementing locks on distributed applications can avoid inventory oversold problems and nonsequential access. The following describes how to implement locks on distributed applications with Redis.

Prerequisites

- You have purchased a Windows ECS.
- You have installed **JDK1.8** (or later) and **Eclipse** on the ECS, and downloaded the **Jedis client**.
- You have purchased a DCS instance and configured the same VPC, subnet, and security group for the DCS instance and the ECS.

Procedure

Step 1 Run Eclipse on the ECS and create a Java project. Then, create a distributed lock implementation class **DistributedLock.java** and a test class **CaseTest.java** for the example code, and reference the Jedis client as a library to the project.

Example content of **DistributedLock.java**:

```
package dcsDemo01;

import java.util.UUID;

import redis.clients.jedis.Jedis;

public class DistributedLock {
    private final String host = "192.168.0.220";
    private final int port = 6379;

    private static final String SUCCESS = "OK";
    private static final String SET_IF_NOT_EXIST = "NX";
    private static final String EXPIRE_TIME = "PX";

    public  DistributedLock(){}

    /*
     * @param lockName       Lock name
     * @param timeout        Timeout for acquiring locks
     * @param lockTimeout    Validity period of locks
     * @return               Lock ID
     */
    public String getLockWithTimeout(String lockName, long timeout, long lockTimeout) {
        String ret = null;
        Jedis jedisClient = new Jedis(host, port);

        try {
            String authMsg = jedisClient.auth("passwd");
            if (!SUCCESS.equals(authMsg)) {
                System.out.println("AUTH FAILED: " + authMsg);
            }

            String identifier = UUID.randomUUID().toString();
            String lockKey = "DLock:" + lockName;
            long end = System.currentTimeMillis() + timeout;

            while(System.currentTimeMillis() < end) {
                String result = jedisClient.set(lockKey, identifier, SET_IF_NOT_EXIST, EXPIRE_TIME, lockTimeout);
                if(SUCCESS.equals(result)) {
                    ret = identifier;
                    break;
                }

                try {
                    Thread.sleep(2);
```

 Copyright © Huawei Technologies Co., Ltd.

```
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }finally {
            jedisClient.quit();
            jedisClient.close();
        }

        return ret;
    }

    /*
     * @param lockName       Lock name
     * @param identifier    Lock ID
     */
    public void releaseLock(String lockName, String identifier) {
        Jedis jedisClient = new Jedis(host, port);

        try {
            String authMsg = jedisClient.auth("passwd");
            if (!SUCCESS.equals(authMsg)) {
                System.out.println("AUTH FAILED: " + authMsg);
            }

            String lockKey = "DLock:" + lockName;
            if(identifier.equals(jedisClient.get(lockKey))) {
                jedisClient.del(lockKey);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }finally {
            jedisClient.quit();
            jedisClient.close();
        }
    }
}
```

NOTICE

The code only shows how DCS implements access control using locks. During actual implementation, deadlock and lock check also need to be considered.

Assume that 20 threads are used to seckill ten Mate 10 mobile phones. The content of **CaseTest.java** is as follows:

```
package dcsDemo01;
import java.util.UUID;

public class CaseTest {
    public static void main(String[] args) {
        ServiceOrder service = new ServiceOrder();
        for (int i = 0; i < 20; i++) {
            ThreadBuy client = new ThreadBuy(service);
            client.start();
        }
    }
}

class ServiceOrder {
    private final int MAX = 10;

    DistributedLock DLock = new DistributedLock();
```

 Copyright © Huawei Technologies Co., Ltd.

```
    int n = 10;

    public void handleOder() {
        String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
        String identifier = DLock.getLockWithTimeout("Huawei Mate 10", 10000, 2000);
        System.out.println("Processing order for user " + userName + "");
        if(n > 0) {
            int num = MAX - n + 1;
            System.out.println("User "+ userName + " is allocated number " + num + " mobile phone. Number
of mobile phones left: " + (--n) + "");
        }else {
            System.out.println("User "+ userName + " order failed. All mobile phones sold out.");
        }
        DLock.releaseLock("Huawei Mate 10", identifier);
    }
}

class ThreadBuy extends Thread {
    private ServiceOrder service;

    public ThreadBuy(ServiceOrder service) {
        this.service = service;
    }

    @Override
    public void start() {
        service.handleOder();
    }
}
```

Step 2 Configure the connection address, port number, and password of the DCS instance in the example code file **DistributedLock.java**.

In **DistributedLock.java**, set **host** and **port** to the connection address and port number of the instance. In the **getLockWithTimeout** and **releaseLock** methods, set **passwd** to the instance access password.

Step 3 Comment out the lock part in the test class **CaseTest**. The following is an example:

```
//The lock code is commented out in the test class:
public void handleOder() {
    String userName = UUID.randomUUID().toString().substring(0,8) + Thread.currentThread().getName();
    //Lock code
    //String identifier = DLock.getLockWithTimeout("Huawei Mate 10", 10000, 2000);
    System.out.println("Processing order for user " + userName + "");
    if(n > 0) {
        int num = MAX - n + 1;
        System.out.println("User "+ userName + " is allocated number " + num + " mobile phone. Number of
mobile phones left: " + (--n) + "")
    }else {
        System.out.println("User "+ userName + " order failed. All mobile phones sold out.");
    }
    //Lock code
    //DLock.releaseLock("Huawei Mate 10", identifier);
}
```

Step 4 Compile and run a lock-free class. The purchases are disordered, as shown in the following:

```
Processing order for user e04934ddThread-5
Processing order for user a4554180Thread-0
User a4554180Thread-0 is allocated number 2 mobile phone. Number of mobile phones left: 8.
Processing order for user b58eb811Thread-10
User b58eb811Thread-10 is allocated number 3 mobile phone. Number of mobile phones left: 7.
Processing order for user e8391c0eThread-19
Processing order for user 21fd133aThread-13
Processing order for user 1dd04ff4Thread-6
```

 Copyright © Huawei Technologies Co., Ltd.

```
User 1dd04ff4Thread-6 is allocated number 6 mobile phone. Number of mobile phones left: 4.
Processing order for user e5977112Thread-3
Processing order for user 4d7a8a2bThread-4
User e5977112Thread-3 is allocated number 7 mobile phone. Number of mobile phones left: 3.
Processing order for user 18967410Thread-15
User 18967410Thread-15 is allocated number 9 mobile phone. Number of mobile phones left: 1.
Processing order for user e4f51568Thread-14
User 21fd133aThread-13 is allocated number 5 mobile phone. Number of mobile phones left: 5.
User e8391c0eThread-19 is allocated number 4 mobile phone. Number of mobile phones left: 6.
Processing order for user d895d3f1Thread-12
User d895d3f1Thread-12 order failed. All mobile phones sold out.
Processing order for user 7b8d2526Thread-11
User 7b8d2526Thread-11 order failed. All mobile phones sold out.
Processing order for user d7ca1779Thread-8
User d7ca1779Thread-8 order failed. All mobile phones sold out.
Processing order for user 74fca0ecThread-1
User 74fca0ecThread-1 order failed. All mobile phones sold out.
User e04934ddThread-5 is allocated number 1 mobile phone. Number of mobile phones left: 9.
User e4f51568Thread-14 is allocated number 10 mobile phone. Number of mobile phones left: 0.
Processing order for user aae76a83Thread-7
User aae76a83Thread-7 order failed. All mobile phones sold out.
Processing order for user c638d2cfThread-2
User c638d2cfThread-2 order failed. All mobile phones sold out.
Processing order for user 2de29a4eThread-17
User 2de29a4eThread-17 order failed. All mobile phones sold out.
Processing order for user 40a46ba0Thread-18
User 40a46ba0Thread-18 order failed. All mobile phones sold out.
Processing order for user 211fd9c7Thread-9
User 211fd9c7Thread-9 order failed. All mobile phones sold out.
Processing order for user 911b83fcThread-16
User 911b83fcThread-16 order failed. All mobile phones sold out.
User 4d7a8a2bThread-4 is allocated number 8 mobile phone. Number of mobile phones left: 2.
```

Step 5 Add the lock code back to **CaseTest**, and compile and run the code. The following shows sequential purchases:

```
Processing order for user eee56fb7Thread-16
User eee56fb7Thread-16 is allocated number 1 mobile phone. Number of mobile phones left: 9.
Processing order for user d6521816Thread-2
User d6521816Thread-2 is allocated number 2 mobile phone. Number of mobile phones left: 8.
Processing order for user d7b3b983Thread-19
User d7b3b983Thread-19 is allocated number 3 mobile phone. Number of mobile phones left: 7.
Processing order for user 36a6b97aThread-15
User 36a6b97aThread-15 is allocated number 4 mobile phone. Number of mobile phones left: 6.
Processing order for user 9a973456Thread-1
User 9a973456Thread-1 is allocated number 5 mobile phone. Number of mobile phones left: 5.
Processing order for user 03f1de9aThread-14
User 03f1de9aThread-14 is allocated number 6 mobile phone. Number of mobile phones left: 4.
Processing order for user 2c315ee6Thread-11
User 2c315ee6Thread-11 is allocated number 7 mobile phone. Number of mobile phones left: 3.
Processing order for user 2b03b7c0Thread-12
User 2b03b7c0Thread-12 is allocated number 8 mobile phone. Number of mobile phones left: 2.
Processing order for user 75f25749Thread-0
User 75f25749Thread-0 is allocated number 9 mobile phone. Number of mobile phones left: 1.
Processing order for user 26c71db5Thread-18
User 26c71db5Thread-18 is allocated number 10 mobile phone. Number of mobile phones left: 0.
Processing order for user c32654dbThread-17
User c32654dbThread-17 order failed. All mobile phones sold out.
Processing order for user df94370aThread-7
User df94370aThread-7 order failed. All mobile phones sold out.
Processing order for user 0af94cddThread-5
User 0af94cddThread-5 order failed. All mobile phones sold out.
Processing order for user e52428a4Thread-13
User e52428a4Thread-13 order failed. All mobile phones sold out.
Processing order for user 46f91208Thread-10
User 46f91208Thread-10 order failed. All mobile phones sold out.
Processing order for user e0ca87bbThread-9
User e0ca87bbThread-9 order failed. All mobile phones sold out.
Processing order for user f385af9aThread-8
User f385af9aThread-8 order failed. All mobile phones sold out.
```

 Copyright © Huawei Technologies Co., Ltd.

```
Processing order for user 46c5f498Thread-6
User 46c5f498Thread-6 order failed. All mobile phones sold out.
Processing order for user 935e0f50Thread-3
User 935e0f50Thread-3 order failed. All mobile phones sold out.
Processing order for user d3eaae29Thread-4
User d3eaae29Thread-4 order failed. All mobile phones sold out.
```

----End

 Copyright © Huawei Technologies Co., Ltd.

2 Ranking with Redis

The best practice for HUAWEI CLOUD DCS guides you through ranking using DCS.

Scenario

Ranking is a function commonly used on web pages and apps. It is implemented by listing key-values in descending order. However, a huge number of concurrent operation and query requests can result in a performance bottleneck, significantly increasing latency.

Ranking using DCS for Redis provides the following advantages:

- Data is stored in the cache, so read/write is fast.
- Multiple types of data structures, such as strings, lists, sets, and hashes are supported.

Operation Guidance

Step 1 Prepare an ECS that runs a Windows OS.

Step 2 Install **JDK1.8** (or later) and **Eclipse** on the ECS, and download the **Jedis client**.

Step 3 Purchase a DCS instance on the HUAWEI CLOUD DCS console. Ensure that you configure the same VPC, subnet, and security group for the DCS instance and the ECS.

Step 4 Run Eclipse on the ECS and create a Java project. Then, create a **productSalesRankDemo.java** file for the example code, and reference the Jedis client as a library to the project.

Step 5 Configure the connection address, port number, and password for the DCS instance in the example code file.

Step 6 Compile and run the code.

----End

Sample Code

```
package dcsDemo02;

import java.util.ArrayList;
```

```
import java.util.List;
import java.util.Set;
import java.util.UUID;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;

public class productSalesRankDemo {
    static final int PRODUCT_KINDS = 30;

    public static void main(String[] args) {
        //Instance connection address, which is obtained from the DCS console.
        String host = "192.168.0.246";
        //Redis port number
        int port = 6379;

        Jedis jedisClient = new Jedis(host, port);

        try {
            //Instance password
            String authMsg = jedisClient.auth("******");
            if (!authMsg.equals("OK")) {
                System.out.println("AUTH FAILED: " + authMsg);
            }

            //Key
            String key = ""Best-seller Rankings";

            jedisClient.del(key);

            //Generate product data at random
            List<String> productList = new ArrayList<>();
            for(int i = 0; i < PRODUCT_KINDS; i ++) {
                productList.add("product-" + UUID.randomUUID().toString());
            }

           //Generate sales volume at random
            for(int i = 0; i < productList.size(); i ++) {
                int sales = (int)(Math.random() * 20000);
                String product = productList.get(i);
               //Insert sales volume into Redis SortedSet
                jedisClient.zadd(key, sales, product);
            }

            System.out.println();
            System.out.println("                  "+key);

           //Obtain all lists and display the lists by sales volume
            Set<Tuple> sortedProductList = jedisClient.zrevrangeWithScores(key, 0, -1);
            for(Tuple product : sortedProductList) {
               System.out.println("Product ID: " + product.getElement() + ", Sales volume: "
                       + Double.valueOf(product.getScore()).intValue());
            }

            System.out.println();
            System.out.println("                  "+key);
            System.out.println("                  Top 5 Best-sellers");

            //Obtain the top 5 best-selling products and display the products by sales volume
            Set<Tuple> sortedTopList = jedisClient.zrevrangeWithScores(key, 0, 4);
            for(Tuple product : sortedTopList) {
               System.out.println("Product ID: " + product.getElement() + ", Sales volume: "
                       + Double.valueOf(product.getScore()).intValue());
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
```

 Copyright © Huawei Technologies Co., Ltd.

```
            jedisClient.quit();
            jedisClient.close();
        }
    }

}
```

Operation Result

Compile and run the preceding Demo code. The operation result is as follows:

```
Best-seller Rankings
Product ID: product-b290c0d4-e919-4266-8eb5-7ab84b19862d, Sales volume: 18433
Product ID: product-e61a0642-d34f-46f4-a720-ee35940a5e7f, Sales volume: 18334
Product ID: product-ceeab7c3-69a7-4994-afc6-41b7bc463d44, Sales volume: 18196
Product ID: product-f2bdc549-8b3e-4db1-8cd4-a2ddef4f5d97, Sales volume: 17870
Product ID: product-f50ca2de-7fa4-45a3-bf32-23d34ac15a41, Sales volume: 17842
Product ID: product-d0c364e0-66ec-48a8-9ac9-4fb58adfd033, Sales volume: 17782
Product ID: product-5e406bbf-47c7-44a9-965e-e1e9b62ed1cc, Sales volume: 17093
Product ID: product-0c4d31ee-bb15-4c88-b319-a69f74e3c493, Sales volume: 16432
Product ID: product-a986e3a4-4023-4e00-8104-db97e459f958, Sales volume: 16380
Product ID: product-a3ac9738-bed2-4a9c-b96a-d8511ae7f03a, Sales volume: 15305
Product ID: product-6b8ad4b7-e134-480f-b3ae-3d35d242cb53, Sales volume: 14534
Product ID: product-26a9b41b-96b1-4de0-932b-f78d95d55b2d, Sales volume: 11417
Product ID: product-1f043255-a1f9-40a0-b48b-f40a81d07e0e, Sales volume: 10875
Product ID: product-c8fee24c-d601-4e0e-9d18-046a65e59835, Sales volume: 10521
Product ID: product-5869622b-1894-4702-b750-d76ff4b29163, Sales volume: 10271
Product ID: product-ff0317d2-d7be-4021-9d25-1f997d622768, Sales volume: 9909
Product ID: product-da254e81-6dec-4c76-928d-9a879a11ed8d, Sales volume: 9504
Product ID: product-fa976c02-b175-4e82-b53a-8c0df96fe877, Sales volume: 8630
Product ID: product-0624a180-4914-46b9-84d0-9dfbbdaa0da2, Sales volume: 8405
Product ID: product-d0079955-eaea-47b2-845f-5ff05a110a70, Sales volume: 7930
Product ID: product-a53145ef-1db9-4c4d-a029-9324e7f728fe, Sales volume: 7429
Product ID: product-9b1a1fd1-7c3b-4ae8-9fd3-ab6a0bf71cae, Sales volume: 5944
Product ID: product-cf894aee-c1cb-425e-a644-87ff06485eb7, Sales volume: 5252
Product ID: product-8bd78ba8-f2c4-4e5e-b393-60aa738eceae, Sales volume: 4903
Product ID: product-89b64402-c624-4cf1-8532-ae1b4ec4cabc, Sales volume: 4527
Product ID: product-98b85168-9226-43d9-b3cf-ef84e1c3d75f, Sales volume: 3095
Product ID: product-0dda314f-22a7-464b-ab8c-2f8f00823a39, Sales volume: 2425
Product ID: product-de7eb085-9435-4924-b6fa-9e9fe552d5a7, Sales volume: 1694
Product ID: product-9beadc07-aab0-438c-ac5e-bcc72b9d9c36, Sales volume: 1135
Product ID: product-43834316-4aca-4fb2-8d2d-c768513015c5, Sales volume: 256

        Best-seller Rankings
        Top 5 Best-sellers
Product ID: product-b290c0d4-e919-4266-8eb5-7ab84b19862d, Sales volume: 18433
Product ID: product-e61a0642-d34f-46f4-a720-ee35940a5e7f, Sales volume: 18334
Product ID: product-ceeab7c3-69a7-4994-afc6-41b7bc463d44, Sales volume: 18196
Product ID: product-f2bdc549-8b3e-4db1-8cd4-a2ddef4f5d97, Sales volume: 17870
Product ID: product-f50ca2de-7fa4-45a3-bf32-23d34ac15a41, Sales volume: 17842
```

 Copyright © Huawei Technologies Co., Ltd.

3 Suggestions on Using Redis

This section provides suggestions on using Redis, including setting key names and values, and using commands.

Key

Suggestions on setting key names:

1. Keep the format consistent.

   Normally, use the service name or database name as the prefix, followed by colons (:). For example: *service name:table name:ID*.

2. Minimize the key length.

   Minimize the key length without compromising clarity of the meaning. Abbreviate common words. For example, **user** can be abbreviated to **u**, and **messages** can be abbreviated to **msg**.

3. Do not include special characters.

Value

Suggestions on setting key values:

1. Avoid using big keys.

   Big keys can result in traffic storms and slow queries. Keep the size of Strings within 10 KB and the number of Hashes, Lists, Sets, or Zsets within 5000.

2. Use appropriate data types.

   For example, to store multiple attributes of a user, you can use multiple keys, such as **set u:1:name "X"** and **set u:1:age 20**. To save memory usage, you can also use the **HMSET** command to set multiple fields to their respective values in the hash stored at one key.

3. Set appropriate timeout.

   Avoid expiring a large number of keys all at once.

Redis Commands

Suggestions on using Redis commands:

 Copyright © Huawei Technologies Co., Ltd.

1. Pay attention to the value of N for commands whose time complexity is O(N).

   For example, the **HGETALL**, **LRANGE**, **SMEMBERS**, **ZRANGE**, and **SINTER** commands may compromise performance if they involve a large number of elements. Alternatively, you can use **SCAN** sister commands, such as **HSCAN**, **SSCAN**, and **ZSCAN** commands.

2. Some commands are not supported in DCS. For details, see **Redis Command Compatibility** and **Web CLI Command Compatibility**.

3. Exercise caution when using the **SELECT** command.

   Redis does not have a strong support for the shared database functionality. Redis is single-threaded, so shared databases interferes with each other. You are advised to use multiple Redis instances instead of using shared databases for one instance. Cluster DCS instances on HUAWEI CLOUD do not support shared databases.

4. For batch operations, use the **MGET** command, **MSET** command, or pipelining to improve the efficiency, but do not include a large number of elements in one batch operation.

   MGET command, **MSET** command, and pipelining differ in the following ways:

   - **MGET** and **MSET** are atomic operations, while pipelining is not.
   - Pipelining can be used to send multiple commands at once, while **MGET** and **MSET** cannot.
   - Pipelining must be supported by both the server and the client.

5. The timeout of Lua scripts is 5s, so avoid using long scripts, such as including time-consuming sleep statements or long loops.

6. When invoking the Lua script, do not use random functions to specify keys. Otherwise, the execution results will be inconsistent between the master and standby nodes, causing data inconsistency.

7. The restrictions on using Lua scripts for cluster instances are as follows:

   - When the **EVAL** or **EVALSHA** command is run, the command parameter must contain at least one key. Otherwise, the client displays the error message " ERR eval/evalsha numkeys must be bigger than zero in redis cluster mode".
   - When the **EVAL** or **EVALSHA** command is run, a cluster DCS Redis instance uses the first key to compute slots. Ensure that the keys to be operated are in the same slot.

Others

1. Use connection pools and persistent connections ("pconnect" in Redis terminology) to control connections and improve the efficiency.

2. Set maxmemory-policy (the eviction policy applied when the maxmemory limit is reached) to one of the following 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.

 Copyright © Huawei Technologies Co., Ltd.

- **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.

3. When deleting big keys, do not always use the **DEL** command.
   - To delete big Hashes, use **HSCAN** + **HDEL** commands.
   - To delete big Lists, use the **LTRIM** command.
   - To delete big Sets, use **SSCAN** + **SREM** commands.
   - To delete big Sorted Sets, use **ZSCAN** + **ZREM** commands.
4. Do not include a large number of commands in a pipeline. A maximum of 1024 commands is recommended for a pipeline of a cluster instance.

 Copyright © Huawei Technologies Co., Ltd.

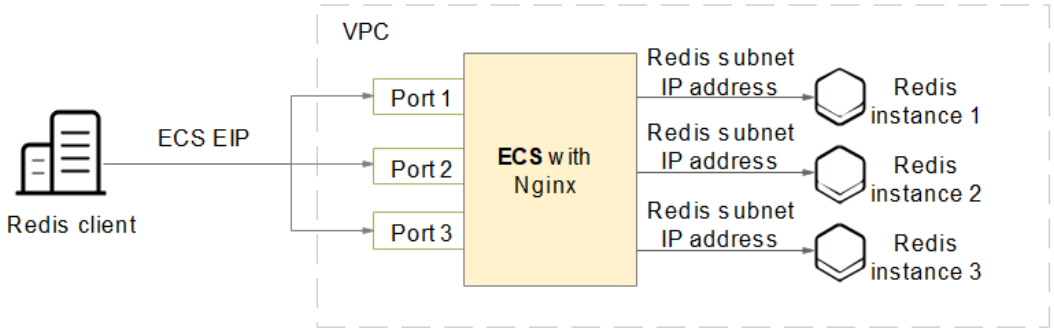
4 Using Nginx for Public Access to Single-node, Master/Standby, or Proxy Cluster DCS Redis 4.0 or 5.0 Instances

Currently, DCS Redis 4.0 or 5.0 instances cannot be bound with elastic IP addresses (EIPs) and cannot be accessed over public networks directly.

This section describes how to access a single-node, master/standby, read/write splitting, or Proxy Cluster DCS Redis 4.0 or 5.0 instance by using a jump server. This solution cannot be used to access a Redis Cluster DCS Redis 4.0 or 5.0 instances over public networks.

As shown in **Figure 4-1**, the ECS where Nginx is installed is a jump server. The ECS is in the same VPC as the DCS Redis instances and can access the DCS Redis instances through the subnet IP addresses. After an EIP is bound to the ECS, the ECS can be accessed over the public network. Nginx can listen on multiple ports and forward requests to different DCS Redis instances.

Figure 4-1 Accessing DCS Redis instances in a VPC by using Nginx



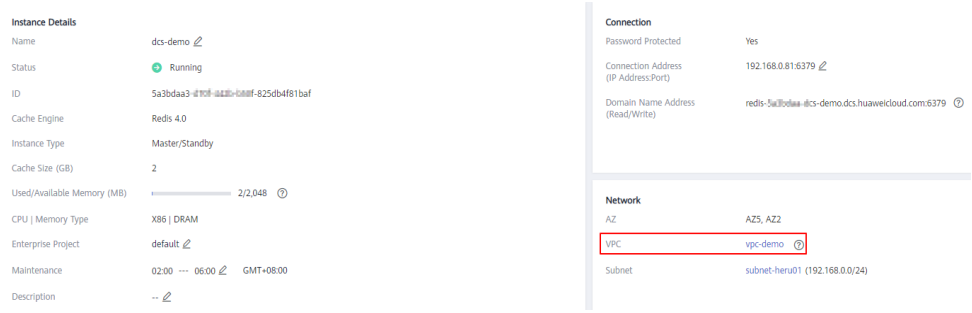
Buying an ECS

Step 1 Obtain the VPC where the DCS Redis instance is deployed.

As shown in the following figure, the master/standby DCS Redis 4.0 or 5.0 instance is deployed in the **vpc-demo** VPC.

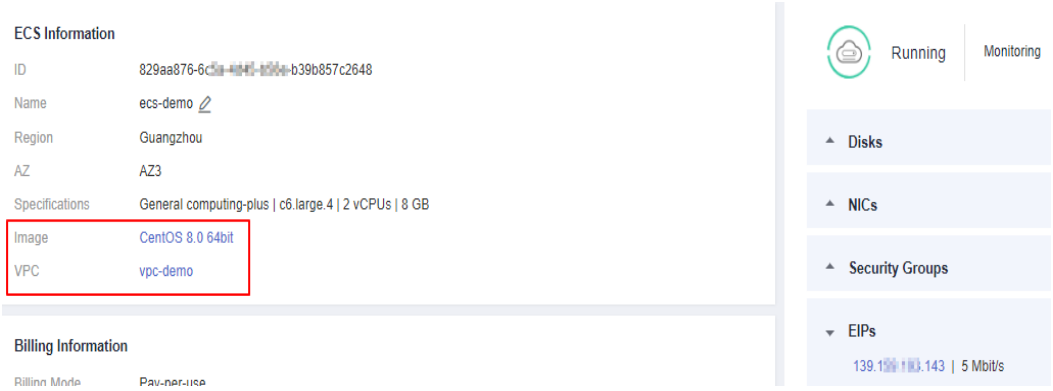
 Copyright © Huawei Technologies Co., Ltd.

Figure 4-2 DCS Redis instance details



Step 2 Buy an ECS. Configure the ECS with the **vpc-demo** VPC, bind an EIP to the ECS, and select the bandwidth as required.

Figure 4-3 ECS details



----End

Installing Nginx

After buying an ECS, install Nginx on the ECS. The following uses CentOS 7.x as an example to describe how to install Nginx. The commands vary depending on the OS.

Step 1 Run the following command to add Nginx to the Yum repository:

sudo rpm -Uvh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm

Step 2 Run the following command to check whether Nginx has been added successfully:

yum search nginx

Step 3 Run the following command to install Nginx:

sudo yum install -y nginx

Step 4 Run the following command to install the stream module:

yum install nginx-mod-stream

Step 5 Run the following commands to start Nginx and set it to run automatically upon system startup:

sudo systemctl start nginx.service

 Copyright © Huawei Technologies Co., Ltd.

sudo systemctl enable nginx.service

Step 6 In the address box of a browser, enter the server address (the EIP of the ECS) to check whether Nginx is installed successfully.

If the following page is displayed, Nginx has been installed successfully.



----End

Setting Up Nginx

After installing Nginx, configure request forwarding rules to specify the ports that Nginx listens on and the DCS Redis instances that Nginx forwards requests to.

Step 1 Run the following commands to modify the configuration file:

cd /etc/nginx

vi nginx.conf

The following is a configuration example. To access multiple DCS Redis instances over public networks, configure multiple **server** sections and configure the DCS Redis instance connection addresses for **proxy_pass**.

```
stream {
    server {
        listen 8080;
        proxy_pass 192.168.0.5:6379;
    }
    server {
        listen 8081;
        proxy_pass 192.168.0.6:6379;
    }
}
```

NOTE

Set **proxy_pass** to the IP address of the DCS Redis instance in the same VPC. You can obtain the IP address from the **Connection Information** area on the DCS instance details page.

 Copyright © Huawei Technologies Co., Ltd.

Figure 4-4 Setting up Nginx

```
#   * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

stream {
   server {
           listen 8080;
           proxy_pass 192.168.0.5:6379;
     }
   server {
           listen 8081;
           proxy_pass 192.168.0.6:6379;
     }
   }
```

Step 2 Run the following command to restart Nginx:

service nginx restart

Step 3 Run the following command to verify whether Nginx has been started:

netstat -an|grep 808

Figure 4-5 Starting Nginx and verifying the start

```
[root@k[illegible]mo src]# service nginx restart
Redirecting to /bin/systemctl restart nginx.service
[root@k[illegible]emo src]# netstat -an |grep 808
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8081            0.0.0.0:*               LISTEN
unix  2      [ ACC ]     STREAM     LISTENING     18084    /var/lib/sss/pipes/private/sbus-monitor
unix  3      [ ]         STREAM     CONNECTED     18086    /var/lib/sss/pipes/private/sbus-monitor
unix  3      [ ]         STREAM     CONNECTED     18085
```

If Nginx is listening on ports 8080 and 8081, Nginx has been started successfully.

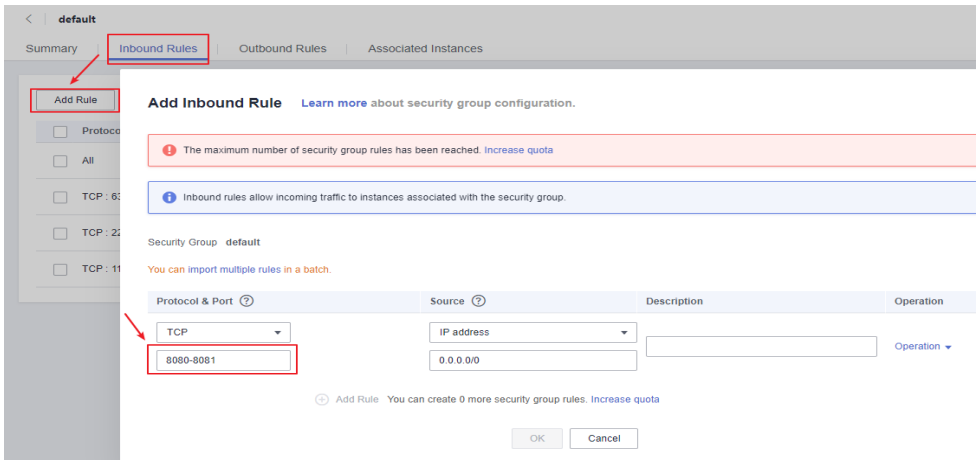
----End

Accessing DCS Redis Instances Using Nginx

Step 1 Log in to the VPC console and check the security group rules of the ECS that servers as the jump server. Ensure that access over ports 8080 and 8081 is allowed.

 Copyright © Huawei Technologies Co., Ltd.

Figure 4-6 Adding an inbound rule for the security group



Step 2 In the public network environment, open the redis-cli and run the following command to check whether the login and query are successful:

NOTE

Ensure that redis-cli has been installed in the public network environment by referring to **redis-cli**.

./redis-cli -h *{myeip}* **-p** *{port}* **-a** *{mypassword}*

In the preceding command, *{myeip}* indicates the host connection address, which should be replaced with the EIP of the ECS. Replace *{port}* with the listening port of Nginx.

As shown in the following figures, the two listening ports are 8080 and 8081, which correspond to two DCS Redis instances.

Figure 4-7 Accessing the first DCS Redis instance using Nginx

```
[root@kafka-demo src]# ./redis-cli -h 121.37.235.247 -p 8080 -a QAZwsx@123
121.37.235.247:8080> set abc 123
OK
121.37.235.247:8080> get abc
"123"
121.37.235.247:8080>
```

Figure 4-8 Accessing the second DCS Redis instance using Nginx

```
[root@kafka-demo src]# ./redis-cli -h 121.37.235.247 -p 8081 -a QAZwsx@123
121.37.235.247:8081> set hello world
OK
121.37.235.247:8081> get hello
"world"
121.37.235.247:8081>
```

The jump server has now been set up. You can access the DCS Redis 4.0 and 5.0 instances over public networks.

----End

 Copyright © Huawei Technologies Co., Ltd.

5 Using SSH Tunneling for Public Access to a DCS Instance

Context

VPCs are used to ensure network security of public cloud services, such as DCS. Your DCS instance can be accessed only by an ECS that is in the same VPC as the instance.

Solution

If an EIP is bound to a HUAWEI CLOUD ECS, you can remotely access the ECS from a local computer.

You can create an SSH tunnel as a proxy to connect your DCS instance and local computer to achieve proxy forwarding.

NOTE

Redis Cluster DCS Redis 4.0 or 5.0 instances do not support public access using this solution.

Prerequisites

You have a DCS instance and a local computer that can connect to the Internet. Tools such as MobaXterm and the Redis client have been installed.

You have an ECS that meets the following requirements:

- The ECS is bound with an EIP for public access.
- The VPC and subnet configured for the ECS are the same as those configured for the DCS instance.
- Security group rules have been correctly configured for the ECS.
- The ECS runs the Linux OS.

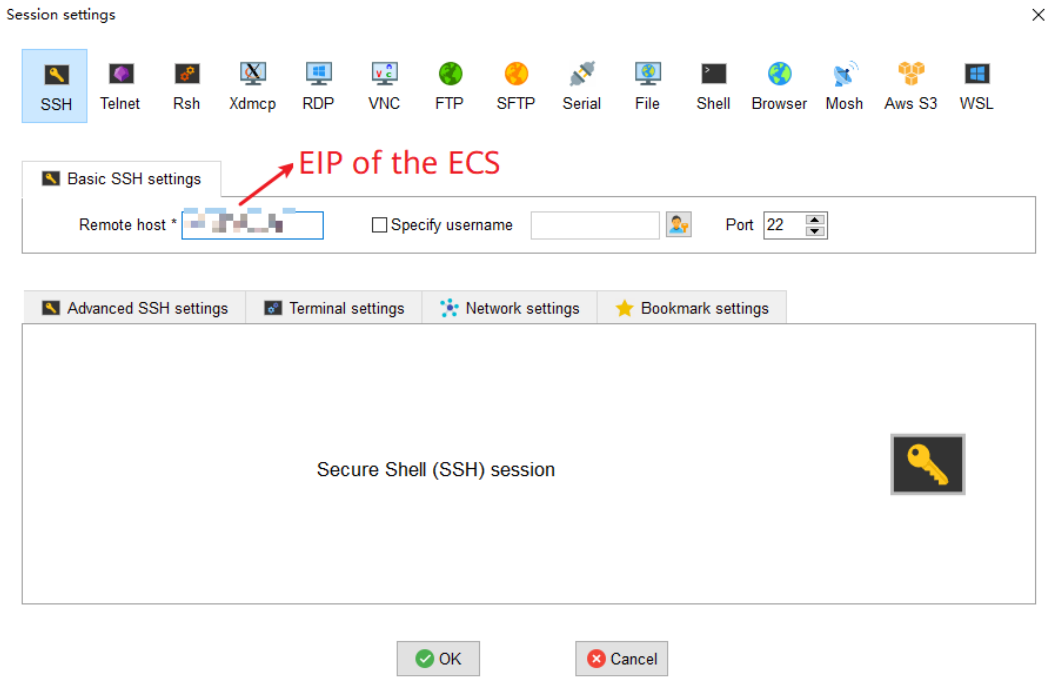
If these prerequisites are met, the ECS can communicate with the DCS instance and you can remotely connect to the ECS using SSH from a local computer.

 Copyright © Huawei Technologies Co., Ltd.

Using MobaXterm to Create a Tunnel as a Jump Server

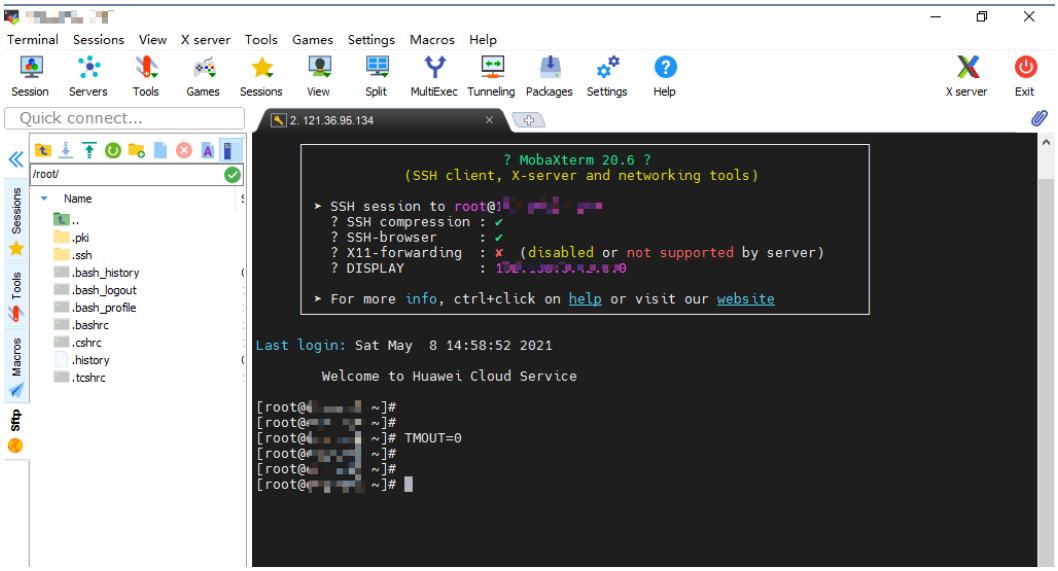
Step 1 Create an SSH session for connecting to the ECS using port 22.

Figure 5-1 Creating an SSH session



Step 2 After the session is configured, enter the username and password to log in to the ECS. After login, enter "TMOUT=0" to prevent the session from being automatically closed due to timeout.

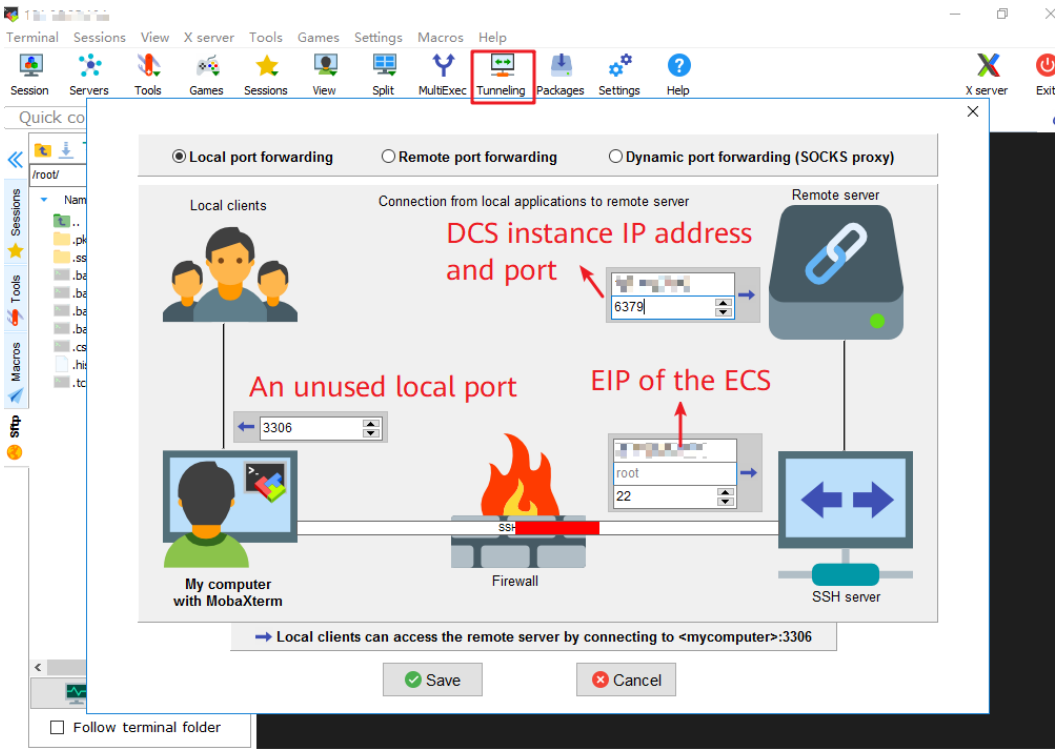
Figure 5-2 Entering "TMOUT=0"



Step 3 Click **Tunneling** to create a tunnel.

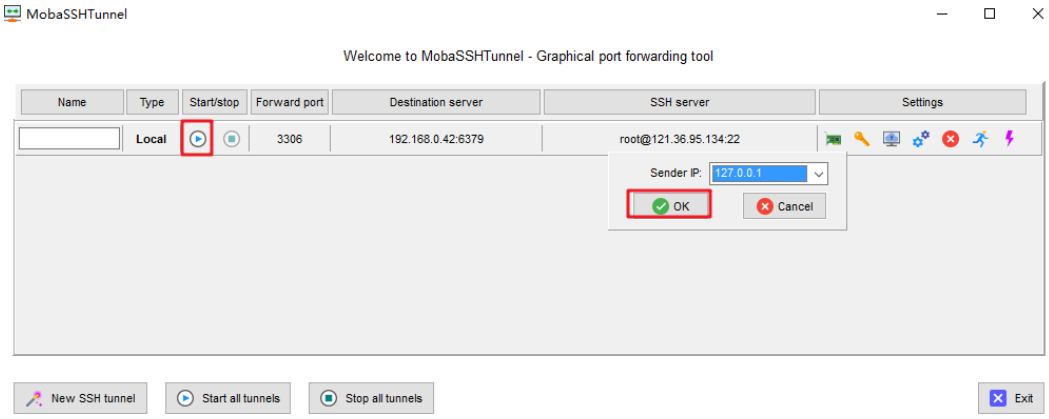
 Copyright © Huawei Technologies Co., Ltd.

Figure 5-3 Creating a tunnel



Step 4 Set the local IP address to 127.0.0.1 and start the tunnel.

Figure 5-4 Starting the tunnel



Step 5 Open the Redis client on the local computer. The following uses the Redis CLI as an example. Run the following command to access the DCS instance:

Redis-cli -h *127.0.0.1* -p *3306* -a *{password}*

Parameter description:

-h *{host name}*: localhost or 127.0.0.1, which is the same as the local IP address configured for the tunnel.

-p *{port number}*: 3306, which is the same as the forward port configured for the tunnel.

-a *{password}*: password of the DCS instance.

 Copyright © Huawei Technologies Co., Ltd.

Step 6 If the connection is successful, the following information is displayed.

Figure 5-5 Successfully connecting to a DCS instance

```
C:\Redis>
C:\Redis>Redis-cli -h 127.0.0.1 -p 3306 -a [redacted]
127.0.0.1:3306> info
# Server
redis_version:5.0.9
patch_version:5.0.9.2
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:0
redis_mode:standalone
os:Linux
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:0.0.0
process_id:1
run_id:74daa94034ce1c8287e3a47b48d446cc04cfdb5b
tcp_port:3397
uptime_in_seconds:2421
```

----End

 Copyright © Huawei Technologies Co., Ltd.

6 Using ELB for Public Access to DCS Across VPCs

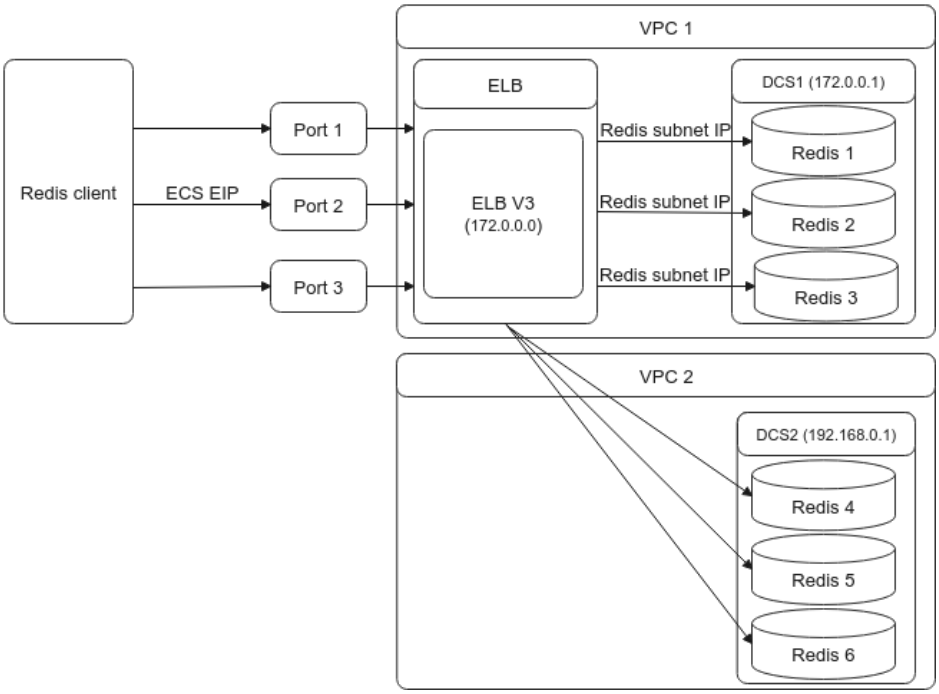
Currently, DCS Redis 4.0 or 5.0 instances cannot be bound with elastic IP addresses (EIPs) and cannot be accessed over public networks directly. This section describes how to use Elastic Load Balancing (ELB) to access a single-node, master/standby, or Proxy Cluster instance or a node in a Redis Cluster instance from the public network across VPCs.

NOTE

Due to cluster node address translation, you cannot access a Redis Cluster as whole. You can only access individual nodes in the cluster.

The following figure shows the process for accessing DCS through ELB.

Figure 6-1 Process for accessing DCS through ELB



 Copyright © Huawei Technologies Co., Ltd.

Configurations

Step 1 **Buy a DCS Redis instance**. Record the IP address and port number of the instance.

Step 2 **Create a dedicated load balancer**.

NOTE

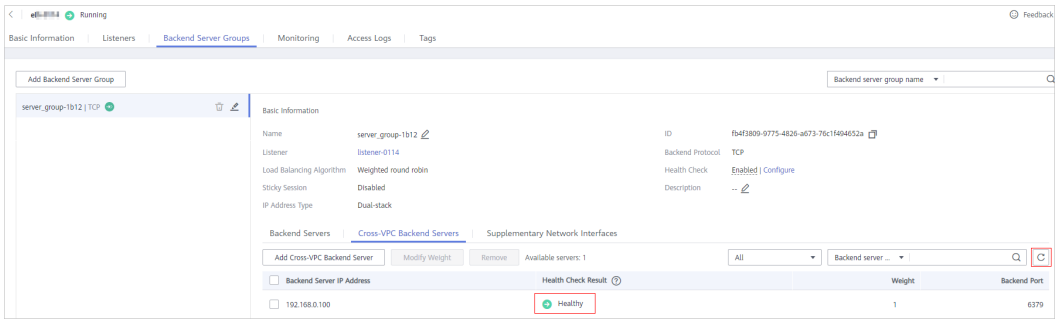
- Public network access is supported by dedicated load balancers and not by shared load balancers.
- For **Specification**, select **Network load balancing (TCP/UDP)**.
- If your DCS Redis instance is not in the same VPC as the load balancer, enable **Cross-VPC Backend** for the load balancer.

Step 3 **Add a TCP listener** to the dedicated load balancer.

NOTE

1. When adding a backend server, click the **Cross-VPC Backend Servers** tab and then click **Add Cross-VPC Backend Server**.
2. In the **Add Cross-VPC Backend Server** dialog box, enter the IP address and port number of your DCS instance.

Step 4 Perform a health check on the IP address of the DCS instance by referring to **Configuring a Health Check for a Dedicated Load Balancer**. If the health check result is **Healthy**, the added cross-VPC backend IP address can be used.



Step 5 **Create a VPC** or use an existing one.

Step 6 **Create a VPC peering connection**. For the local VPC, select the VPC whose CIDR block contains the private IPv4 address of your load balancer. For the peer VPC, select the VPC whose CIDR block contains the IP address of your DCS instance.

NOTE

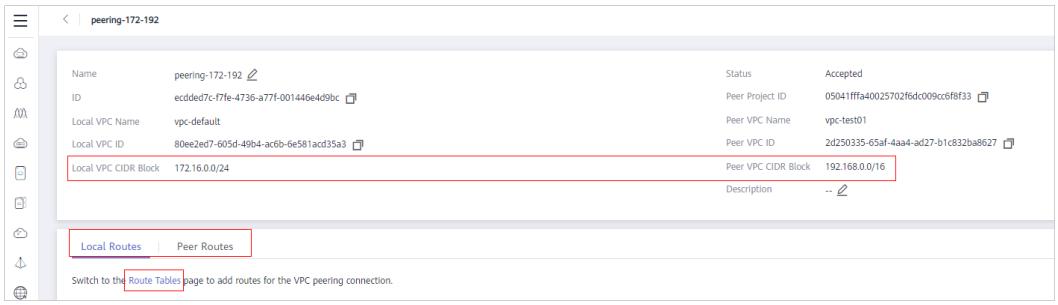
If your load balancer and DCS instance are in the same VPC, skip **Step 6** to **Step 9**.

Step 7 Click the name of the VPC peering connection to go to its details page.



Step 8 Click **Route Tables**.

 Copyright © Huawei Technologies Co., Ltd.



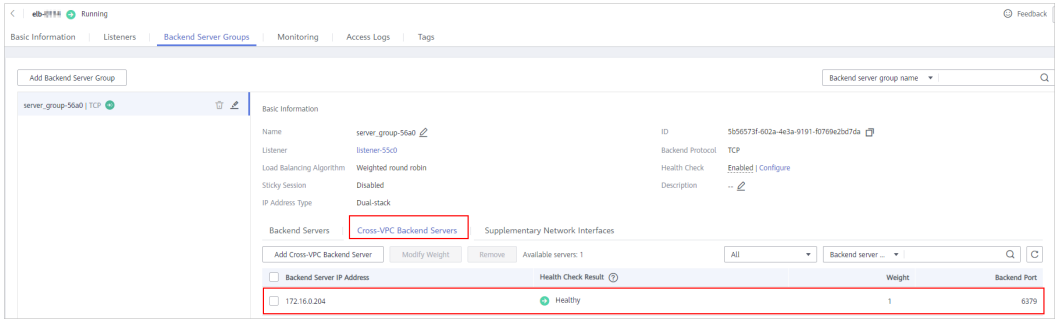
Step 9 Configure local and peer routes for the VPC peering connection.

1. Click **Add Route**. In the displayed **Add Route** dialog box, set **Destination** to the value of **Peer VPC CIDR Block** mentioned in **Step 6**, set **Next Hop Type** to **VPC peering connection**, retain the default setting for **Next Hop**, and click **OK**.
2. Click **Add Route**. In the displayed **Add Route** dialog box, set **Destination** to the value of **Local VPC CIDR Block** mentioned in **Step 6**, set **Next Hop Type** to **VPC peering connection**, retain the default setting for **Next Hop**, and click **OK**.

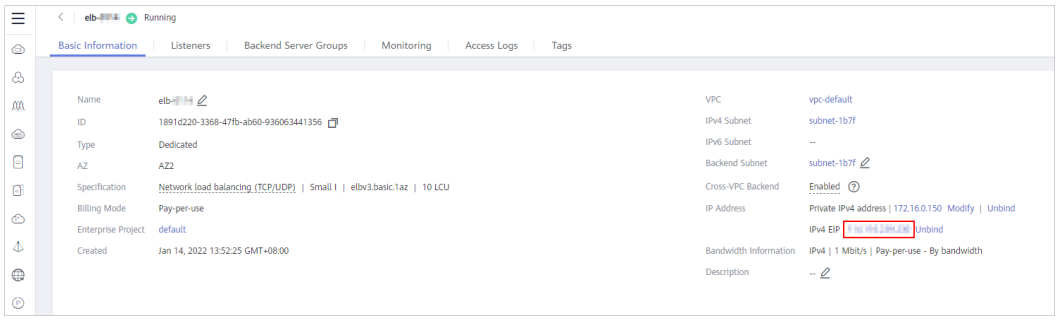
----End

Connecting to the DCS Instance Through ELB

- Connecting to a node in a Redis Cluster DCS instance through ELB
  a. A Redis Cluster contains multiple master/replica pairs. In **Step 3**, you can add the IP address and port number of any master or replica node for the cross-VPC backend server.



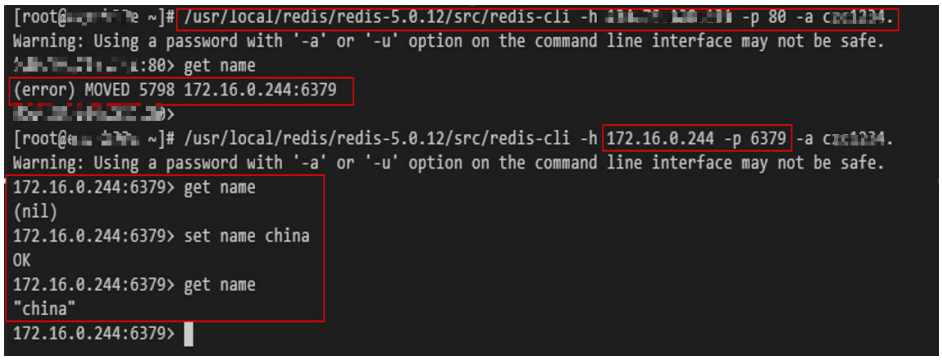
  b. View the basic information of the load balancer created in **Step 2**.



  c. **Buy an ECS**, log in to it, and install the Redis client by referring to **redis-cli**.

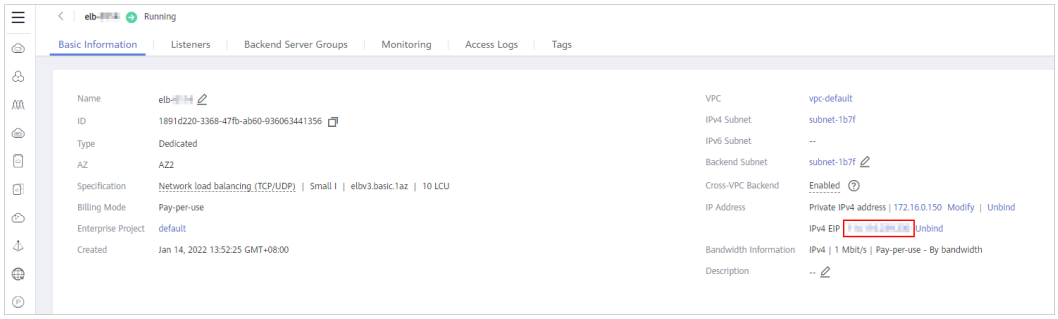
 Copyright © Huawei Technologies Co., Ltd.

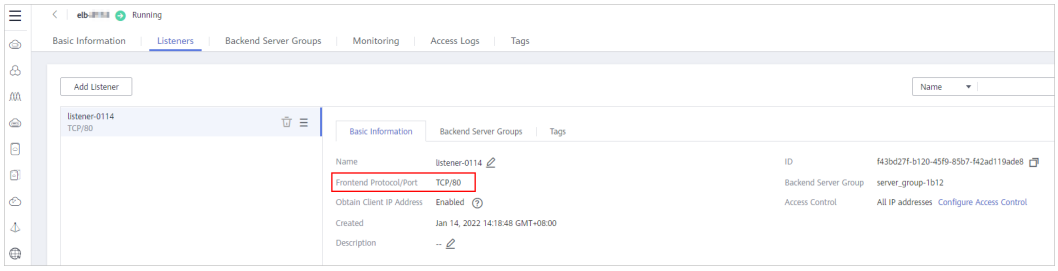
d. On the Redis client, connect to the DCS instance using the IP address and port number configured in **Step 3**. If you use the EIP and port number of the load balancer, an error will be reported.



- Connecting to a single-node, master/standby, or Proxy Cluster DCS instance through ELB

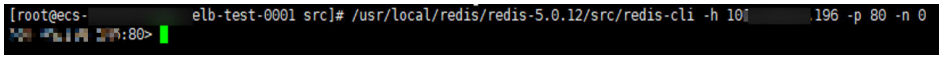
  a. View the IPv4 EIP and port number of the load balancer created in **Step 2**.



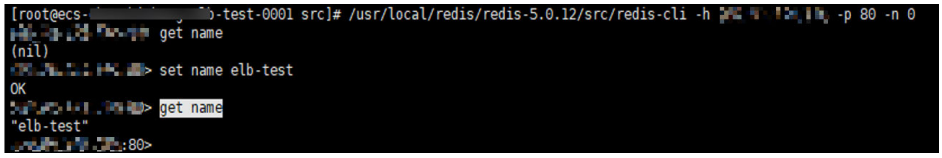


  b. **Buy an ECS**, log in to it, and install the Redis client by referring to **redis-cli**.

  c. Use redis-cli to access the load balancer using its EIP and port number (which is 80).

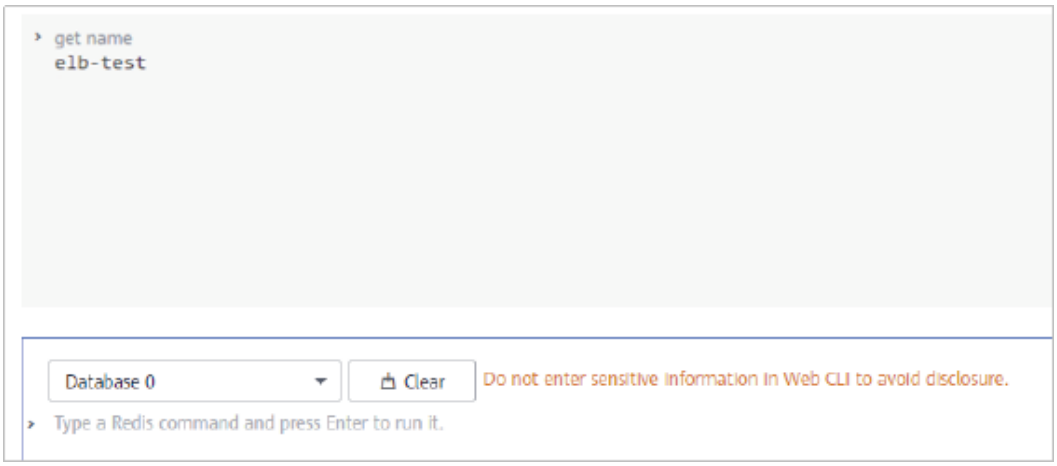


  d. Write a key through ELB.



  e. Log in to the DCS console. On the **Cache Manager** page, choose **More** > **Connect to Redis** in the row that contains the DCS instance created in **Step 1**. Check whether the key written in **d** exists.

 Copyright © Huawei Technologies Co., Ltd.



 Copyright © Huawei Technologies Co., Ltd.