ApsaraDB for Redis

Quick Start

MORE THAN JUST CLOUD | Alibaba Cloud

Quick Start

Purpose of the document

This document describes how to create an ApsaraDB for Redis instance, helping you know the procedures from purchasing an ApsaraDB for Redis instance to using the instance.

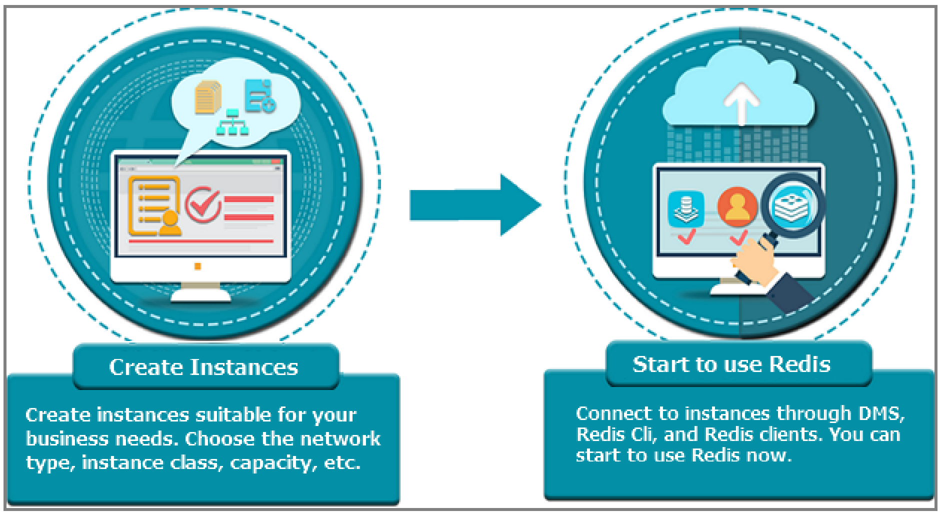
Target reader

- Users purchasing an ApsaraDB for Redis instance for the first time.
- Users who want to know how to connect an ApsaraDB for Redis instance.

Quick start flowchart

If you use ApsaraDB for Redis for the first time, see Limits and About Redis console first.

Generally, you must follow these steps from instance purchasing to instance use.


Create Instances
Create instances suitable for your business needs. Choose the network type, instance class, capacity, etc.
Start to use Redis
Connect to instances through DMS, Redis Cli, and Redis clients. You can start to use Redis now.

The Redis console is a web application that manages ApsaraDB for Redis instances. On the console, you can create and manage instances, set networks and passwords, and perform other operations on the user interface.

Prerequisites

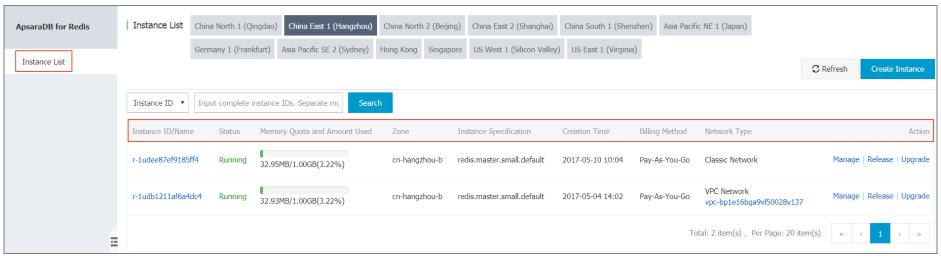
You log on to the Redis console by using your Alibaba Cloud account. If you do not have an Alibaba Cloud account, click register.

Console overview

Console homepage

The console homepage displays the same information for ApsaraDB for Redis instances of all types.

Log on to the **Redis console**, and go to the **Instance List** page, as shown in the following figure. (The figure here is only used for an example. See the actual interface when using this document.)



Instance ID/Name	Status	Memory Quota and Amount Used	Zone	Instance Specification	Creation Time	Billing Method	Network Type	Action
r-1udee87ef9185ff4	Running	32.95MB/1.00GB(3.22%)	cn-hangzhou-b	redis.master.small.default	2017-05-10 10:04	Pay-As-You-Go	Classic Network	Manage | Release | Upgrade
r-1udb1211af6a4dc4	Running	32.93MB/1.00GB(3.22%)	cn-hangzhou-b	redis.master.small.default	2017-05-04 14:02	Pay-As-You-Go	VPC Network vpc-bp1e16bga9vl50028v137	Manage | Release | Upgrade

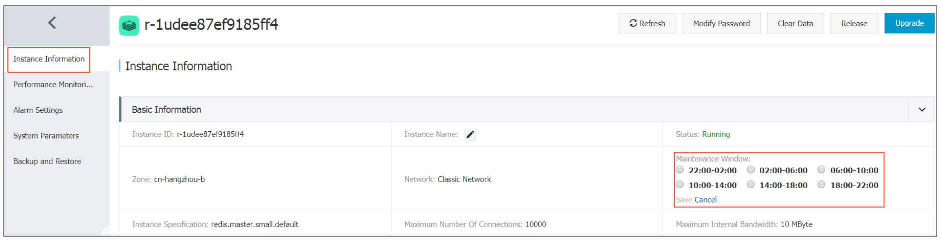
Total: 2 item(s) , Per Page: 20 item(s)

On the **Instance List** page, the following information is displayed: **Instance ID**, **Status**, **Memory Quota and Amount Used**, **Zone**, **Creation Time**, **Billing Method**, and **Network Type**.

> **Note**: **Memory Quota and Amount Used** is offline statistics made by the underlying system based on the collected information. A delay of about 10 minutes always exists.

Maintenance window

You can modify the O&M period on the **Instance Information** page. Alibaba Cloud maintains the instances during the O&M time, which may result in system flash. We recommend that you set the maintenance window in the idle service hours.

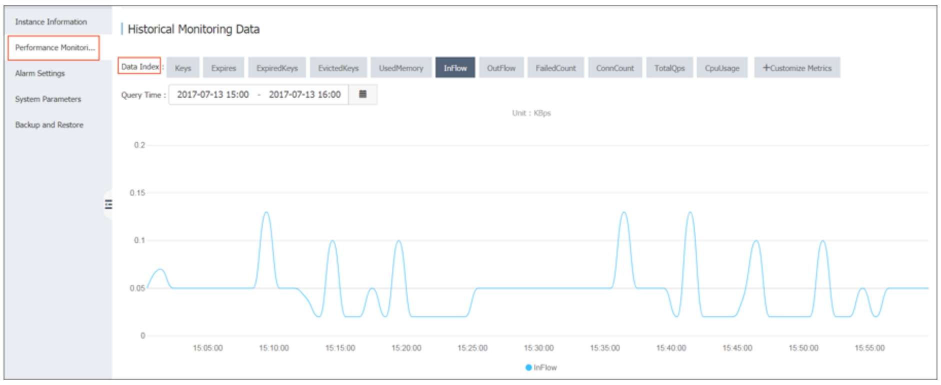


Instance ID: r-1udee87ef9185ff4
Instance Name:
Status: Running
Zone: cn-hangzhou-b
Network: Classic Network
Maintenance Window:
○ 22:00-02:00 ○ 02:00-06:00 ○ 06:00-10:00
○ 10:00-14:00 ○ 14:00-18:00 ○ 18:00-22:00
Save Cancel
Instance Specification: redis.master.small.default
Maximum Number Of Connections: 10000
Maximum Internal Bandwidth: 10 MByte

Performance monitoring

Click **Instance ID** to go to the **Instance Information** page. In the left-side navigation pane, choose

Performance Monitoring to view historical performances of your ApsaraDB for Redis instances. Different metrics are displayed.



Different metrics are displayed after you click **Performance Monitoring**. Metrics of basic monitoring groups are described as follows.

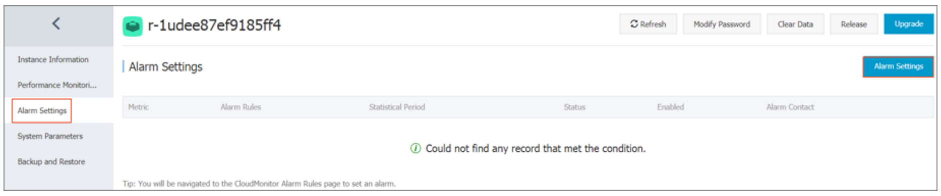
Basic metrics	Description
Keys	Total number of keys of all backend ApsaraDB for Redis instances. Data on all backend nodes of a cluster instance is aggregated.
Expires	Total number of keys for which an expiration time is set.
ExpiredKeys	Number of expired keys. The value is the sum of all expired keys, excluding the number of keys for which an expiration time is set but do not expire currently. Besides, it indicates the number of accumulated expired keys, instead of the number of expired keys in the current time. **Note**: In case of master-slave switchover, the value indicates the number of expired keys in the new master database.
EvictedKeys	Number of evicted keys. The value indicates the sum of keys which are evicted because the memory is used out, instead of the number of keys evicted in the current second. **Note**: In case of master-slave switchover, the value indicates the number of expired keys in the new master database.
UsedMemory	Memory in use currently. When a new instance is created, database metadata is generated. For master-slave instances, the generated database metadata occupies a space of at least 30 MB. For cluster instances, the generated database metadata

	occupies a space of about 30 MB multiplied by the number of nodes. A space of at least 200 MB is occupied.
InFlow	Current traffic per second at the backend ApsaraDB for Redis ingress. The unit is KB/s.
OutFlow	Current traffic per second at the backend ApsaraDB for Redis egress. The unit is KB/s.
ConnCount	Count of current client connections of ApsaraDB for Redis.
FailedCount	It makes no sense for master-slave instances because the client is directly connected to the backend database. For cluster instances, the parameter indicates the number of failed operations from Proxy to ApsaraDB for Redis, including the number of abnormal operations caused by time-out, disconnection, or other exceptions. For some ApsaraDB for Redis of earlier versions, the value is a historical value. In such case, no error is reported when FaileCount is not set. For ApsaraDB for Redis of the new version, the value indicates the statistical value in each second. In the future, the value indicates the statistical value in each second for ApsaraDB for Redis of later versions.
TotalQps	QPS of ApsaraDB for Redis.
CpuUsage	CPU usage of the current ApsaraDB for Redis backend.

Note: You can click **Customize Metrics** to monitor the number of accesses to different operating commands, for example, the number of accesses to the set command per second. For more information, see Performance Monitoring.

Alarm settings

Click **Alarm Settings** in the left-side navigation pane, and click the **Alarm Settings** button to go to the setting page of CloudMonitor.



Could not find any record that met the condition.

Tip: You will be navigated to the CloudMonitor Alarm Rules page to set an alarm.

You can create a metric for ApsaraDB for Redis instances as guided. We recommend that you set a memory metric for all cluster instances to monitor the memory of sub-nodes of the cluster instances.

System parameters

You can set common parameters of ApsaraDB for Redis on the **System Parameters** page, for example, setting an eviction policy and notify-keypsace-events. For more information, see Parameter settings.

Backup and recovery

On the backup and recovery page, you can set a backup and the automatic backup time, and clone an instance. For more information, see Backup and recovery.

Item	Description
List data type	The number of lists is not restricted. The size of single element cannot exceed 512 MB. We recommend that one lists contain no more than 8192 elements, and the maximum value length cannot exceed 1 MB.
Set data type	The number of sets is not restricted. The size of single element cannot exceed 512 MB. We recommend that one set contain no more than 8192 elements, and the maximum value length cannot exceed 1 MB.
SortedSet data type	The number of SortedSets is not restricted. The size of single element cannot exceed 512 MB. We recommend that one SortedSet contain no more than 8192 elements, and the maximum value length cannot exceed 1 MB.
Hash data type	The number of fields is not restricted. The size of single element cannot exceed 512 MB. We recommend that one field contain no more than 8192 elements, and the maximum value length cannot exceed 1 MB.
Restriction on the database number	Each instance supports 256 databases.
Redis commands supported	For more information, see Supported Redis commands.
Monitoring alert	ApsaraDB for Redis does not provide the capacity alert function. You can configure this function on CloudMonitor. For more information, see ApsaraDB for Redis monitoring. We recommend that you set alert for the following metrics: instance fault, instance master-slave switchover, connection usage, failed operation count, capacity usage, write bandwidth usage, and read bandwidth usage.
Expired data deletion policy	- Active expiration: The system periodically detects and deletes expired keys in the

	background. - Passive expiration: The system deletes expired keys when users access keys.
Idle connection recovery mechanism	Idle Refis connection is not automatically recovered by the server, and must be managed by the user.
Data persistence policy	AOF_FSYNC_EVERYSEC is enabled , and fysnc is performed every second.

ApsaraDB for Redis is compatible with Redis 3.0 and supports Redis 3.0 GEO commands. Currently, some commands are temporarily unavailable and restricted.

Supported command operations

Keys	String	Hash	List	Set	SortedSet
DEL	APPEND	HDEL	BLPOP	SADD	ZADD
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD
EXISTS	BITOP	HGET	BRPOPLPUSH	SDIFF	ZCOUNT
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY
EXPIREAT	DECR	HINCRBY	LINSERT	SINTER	ZRANGE
MOVE	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE
PERSIST	GET	HKEYS	LPOP	SISMEMBER	ZRANK
PEXPIRE	GETBIT	HLEN	LPUSH	SMEMBERS	ZREM
PEXPTREAT	GETRANGE	HMGET	LPUSHX	SMOVE	ZREMRANGEBYRANK
PTTL	GETSET	HMSET	LRANGE	SPOP	ZREMRANGEBYSCORE
RANDOMKEY	INCR	HSET	LREM	SRANDMEMBER	ZREVRANGE
RENAME	INCRBY	HSETNX	LSET	SREM	ZREVRANGEBYSCORE
RENAMENX	INCRBYFLOAT	HVALS	LTRIM	SUNION	ZREVRANK
RESTORE	MGET	HSCAN	RPOP	SUNIONSTORE	ZSCORE
SORT	MSET		RPOPLPUSH	SSCAN	ZUNIONSTORE

TTL	MSETNX		RPUSH		ZINTERSTORE
TYPE	PSETEX		RPUSHX		ZSCAN
SCAN	SET				ZRANGEBYLEX
OBJECT	SETBIT				ZLEXCOUNT
	SETEX				ZREMRANGEBYLEX
	SETNX				
	SETRANGE				
	STRLEN				

And

HyperLogLog	**Pub/Sub (publish/subscription)**	**Transaction**	**Connection**	**Server**	**Scripting**	**Geo (geological position)**
PFADD	PSUBSCRIBE	DISCARD	AUTH	FLUSHALL	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	FLUSHDB	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	DBSIZE	SCRIPT EXISTS	GEOPOS
	PUNSUBSCRIBE	UNWATCH	QUIT	TIME	SCRIPT FLUSH	GEODIST
	SUBSCRIBE	WATCH	SELECT	INFO	SCRIPT KILL	GEORADIUS
	UNSUBSCRIBE			KEYS	SCRIPT LOAD	GEORADIUSBYMEMBER
				CLIENT KILL		
				CLIENT LIST		
				CLIENT GETNAME		
				CLIENT SETNAME		
				CONFIG GET		
				MONITO		

				R		
				SLOWLO G		

Commands temporarily unavailable

Keys	Server
MIGRATE	BGREWRITEAOF
	BGSAVE
	CONFIG REWRITE
	CONFIG SET
	CONFIG RESETSTAT
	COMMAND
	COMMAND COUNT
	COMMAND GETKEYS
	COMMAND INFO
	DEBUG OBJECT
	DEBUG SEGFAULT
	LASTSAVE
	ROLE
	SAVE
	SHUTDOWN
	SLAVEOF
	SYNC

Commands restricted for cluster instances

Keys	Strings	Lists	HyperLogLo g	Transaction	Scripting
RENAME	MSETNX	RPOPLPUSH	PFMERGE	DISCARD	EVAL
RENAMENX			PFCOUNT	EXEC	EVALSHA
SORT				MULTI	SCRIPT EXISTS
				UNWATCH	SCRIPT FLUSH

				WATCH	SCRIPT KILL
				WATCH	SCRIPT LOAD

Self-developed commands for cluster instances

info key: Used to query the slot and DB of a key. The native info command of ApsaraDB for Redis can contain only one optional section (info [section]). Currently, some commands are restricted for the cluster instances of ApsaraDB for Redis. Therefore, all keys must be in the same slot. info key allows you to check whether keys are in the same slot. This command is used as follows:

```
 127.0.0.1:6379> info key test_key
slot:15118 db:0
```

iinfo: This command is similar to the info command. It is used to run info on a specified ApsaraDB for Redis node. This command is used as follows:

iinfo db_idx [section]

Note: The value range of db_idx is [0, nodecount), the value of nodecount is obtained by running info, and the value of section is set in the same way as the Redis standard optional parameters for the info command. For more information about how to use the iinfo command on an ApsaraDB for Redis node, see How to view the memory of a sub-instance of an ApsaraDB for Redis cluster.

riinfo: This command is similar to the iinfo command. It can be used only in read/write splitting mode. idx is added to specify the readonly slave on which the info command is run. In a read/write splitting cluster, idx is used to specify the readonly slave on which the info command is run. If idx is used in a non-read/write splitting cluster, an error is returned. This command is used as follows:

riinfo db_idx ro_slave_idx [section]

Notes:

Restricted commands of cluster instances support only scenarios where keys to be operated are evenly distributed in a single hash slot and data of multiple hash slots are not merged. Therefore, you must use the hash tag to make sure that keys to be operated are evenly distributed in one hash slot.

For example, if key1, aakey, and abkey3 are to be operated, you must save them in {key}1, aa{key}, and ab{key}3 modes. In this case, restricted commands can take effect when being called. For more information about how to use the hash tag, see the official documentation of ApsaraDB for Redis at: http://redis.io/topics/cluster-spec.

For more information about ApsaraDB for Redis commands, see the Official documentation.

ApsaraDB for Redis supports Pay-As-You-Go and Subscription instances. The following describes how to purchase a Pay-As-You-Go instance. The precedure is similar for Subscription instances.

Prerequisites

Before activating ApsaraDB for Redis, you must have at least one ECS instance. For more information about how to purchase an ECS instance, see Purchase an ECS instance.

Procedure

Go to ApsaraDB for Redis homepage, and click **Buy Now**. Or you can log on to the Redis console and click **Create Instance** in the upper-right corner.

Choose **Region**, **Zone**, **Instance Type**, **Network Type**, and **Quantity**, and set the **Logon Password** and **Instance Name**.

Note:

- Through configuration change, a master-slave instance can become a cluster instance which has functions different to those of the master-slave instance. For more information, see Commands supported by ApsaraDB for Redis.

  For how to select network type, see Set the network type.

  ApsaraDB for Redis can be accessed only through the intranet. We recommend that you configure ApsaraDB for Redis instance and the ECS instance in the same zone of the same region.

Click **Buy Now** to go to the **Confirm Order** page. Read and accept the Terms of Service for ApsaraDB for Redis, check the order information, and click **Pay Now** to make the payment.

Select a payment method on the payment page and click the **Confirm** button. After you make the payment, a message that reads “Payment Successful” is displayed. After one to five minutes you can log on to the console to view the instance purchased.

> **Note**: ApsaraDB for Redis is consistent with Redis in terms of product behavior. When a new instance is created, it generates database metadata which occupies a fraction of the instance’ s storage space. The occupied space is shown on the ApsaraDB for Redis Console.
>
> For master-slave instances, the generated database metadata occupies a space of about 32 MB.
>
> For cluster instances, the generated database metadata occupies a space of about 32 MB multiplied by the number of nodes.

Connect to Redis

As ApsaraDB for Redis is completely compatible with the native database service, their databases are connected in similar ways. Any clients compatible with the Redis protocol can access Alibaba Cloud ApsaraDB for Redis. You can choose any Redis clients based on their application features.

Note: ApsaraDB for Redis only supports access requests from the Alibaba Cloud intranet rather than those from the Internet. That means only Redis clients installed on ECS instances of the same node can be connected to ApsaraDB for Redis for data operations.

To use Redis clients, see Clients.

- Jedis client
- phpredis client
- redis-py client
- C/C++ client
- .net client
- node-redis client

Jedis client

The Jedis client can access ApsaraDB for Redis through either of the following methods:

Jedis single-connection

JedisPool connection

Procedure

Click download address to download and install the Jedis client.

Example of Jedis single-connection

Open the Eclipse client, create a project, and enter the following code segment:

```
import redis.clients.jedis.Jedis;

public class jedistest {
public static void main(String[] args) {
try {
String host = "xx.kvstore.aliyuncs.com";//The access URL is displayed on the console.
int port = 6379;
Jedis jedis = new Jedis(host, port);
//Authentication information
jedis.auth("password");//password
String key = "redis";
String value = "aliyun-redis";
//Select a database. (The default value is 0.)
jedis.select(1);
//Set a key.
jedis.set(key, value);
System.out.println("Set Key " + key + " Value: " + value);
//Get the key.
String getvalue = jedis.get(key);
System.out.println("Get Key " + key + " ReturnValue: " + getvalue);
jedis.quit();
jedis.close();
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

Run the project. If the following result is output on the Eclipse console, you have successfully connected to ApsaraDB for Redis.

```
Set Key redis Value aliyun-redis
Get Key redis ReturnValue aliyun-redis
```

Then you can use your local Jedis client to operate your ApsaraDB for Redis instance. You can also connect to your ApsaraDB for Redis instance through JedisPool.

Example of JedisPool connection

Open the Eclipse client, create a project, and configure the pom file as follows:

```
<dependency>
<groupId>redis.clients</groupId>
<artifactId>jedis</artifactId>
<version>2.7.2</version>
<type>jar</type>
<scope>compile</scope>
</dependency>
```

Add the following application to the project:

```
import org.apache.commons.pool2.PooledObject;
import org.apache.commons.pool2.PooledObjectFactory;
import org.apache.commons.pool2.impl.DefaultPooledObject;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;

import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;
```

If your Jedis client version is Jedis-2.7.2, enter the following code in the project:

```
JedisPoolConfig config = new JedisPoolConfig();
//Maximum idle connections, which are evaluated by the application. Do not set it to a value
greater than the maximum connections of an ApsaraDB for Redis instance.
config.setMaxIdle(200);
//Maximum connections, which are evaluated by the application. Do not set it to a value
greater than the maximum connections of an ApsaraDB for Redis instance.
config.setMaxTotal(300);
config.setTestOnBorrow(false);
config.setTestOnReturn(false);

String host = "*.aliyuncs.com";
String password = "password";
JedisPool pool = new JedisPool(config, host, 6379, 3000, password);
Jedis jedis = null;
try {
jedis = pool.getResource();
/// ... do stuff here ... for example
jedis.set("foo", "bar");
String foobar = jedis.get("foo");
jedis.zadd("sose", 0, "car");
jedis.zadd("sose", 0, "bike");
Set<String> sose = jedis.zrange("sose", 0, -1);
} finally {
if (jedis != null) {
```

```
jedis.close();
}
}
/// ... when closing your application:
pool.destroy();
```

If your Jedis client version is Jedis-2.6 or Jedis-2.5, enter the following code in the project:

```
JedisPoolConfig config = new JedisPoolConfig();
//Maximum idle connections, which are evaluated by the application. Do not set it to a value greater than the maximum connections of an ApsaraDB for Redis instance.
config.setMaxIdle(200);
//Maximum connections, which are evaluated by the application. Do not set it to a value greater than the maximum connections of an ApsaraDB for Redis instance.
config.setMaxTotal(300);
config.setTestOnBorrow(false);
config.setTestOnReturn(false);
String host = "*.aliyuncs.com";
String password = "password";
JedisPool pool = new JedisPool(config, host, 6379, 3000, password);
Jedis jedis = null;
boolean broken = false;
try {
jedis = pool.getResource();
/// ... do stuff here ... for example
jedis.set("foo", "bar");
String foobar = jedis.get("foo");
jedis.zadd("sose", 0, "car");
jedis.zadd("sose", 0, "bike");
Set<String> sose = jedis.zrange("sose", 0, -1);
} catch(Exception e) {
broken = true;
} finally {
if (broken) {
pool.returnBrokenResource(jedis);
} else if (jedis != null) {
pool.returnResource(jedis);
}
}
```

Run the project. If the following result is output on the Eclipse console, you have successfully connected to ApsaraDB for Redis.

```
Set Key redis Value aliyun-redis
Get Key redis ReturnValue aliyun-redis
```

Then you can use your local Jedis client to operate your ApsaraDB for Redis instance.

phpredis client

Procedure

Click download address to download and install the phpredis client.

In any editor supporting php editing, enter the following code:

```
 <?php
/* Replace the following parameter values with the host of the connected instance and the port number. */
$host = "localhost";
$port = 6379;

/* Replace the following parameter values with the instance ID and instance password. */
$user = "test_username";
$pwd = "test_password";
$redis = new Redis();
if ($redis->connect($host, $port) == false) {
die($redis->getLastError());
}

if ($redis->auth($pwd) == false) {
die($redis->getLastError());
}
/* The database can be operated after authentication. For more information, see
https://github.com/phpredis/phpredis. */
if ($redis->set("foo", "bar") == false) {
die($redis->getLastError());
}
$value = $redis->get("foo");
echo $value;
?>
```

3. Run the preceding code. Then You can use your local phpredis client to access your ApsaraDB for Redis instance. For more information, see https://github.com/phpredis/phpredis.

redis-py client

Procedure

Click download address to download and install the redis-py client.

In any editor supporting Python editing, enter the following code. Then you can use a local redis-py client to connect to and operate the database.

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
import redis

#Replace the following parameter values with the host of the connected instance and the port number.
host = 'localhost'
port = 6379

#Replace the following parameter values with the instance password.
pwd = 'test_password'
r = redis.StrictRedis(host=host, port=port, password=pwd)

#The database can be operated after a connection is created. For more information, see
https://github.com/andymccurdy/redis-py.
r.set('foo', 'bar');
print r.get('foo')
```

C/C++ client

Procedure

Download, compile, and install the C client. The code for compiling and installation is as follows:

```
 git clone https://github.com/redis/hiredis.git
cd hiredis
make
sudo make install
```

Enter the following code in the C/C++ editor:

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
int main(int argc, char **argv) {
unsigned int j;
redisContext *c;
redisReply *reply;
if (argc < 4) {
printf("Usage: example xxx.kvstore.aliyuncs.com 6379 instance_id password\n");
exit(0);
}
const char *hostname = argv[1];
const int port = atoi(argv[2]);
const char *instance_id = argv[3];
const char *password = argv[4];
struct timeval timeout = { 1, 500000 }; // 1.5 seconds
c = redisConnectWithTimeout(hostname, port, timeout);
if (c == NULL || c->err) {
if (c) {
printf("Connection error: %s\n", c->errstr);
redisFree(c);
} else {
printf("Connection error: can't allocate redis context\n");
}
exit(1);
}
/* AUTH */
reply = redisCommand(c, "AUTH %s", password);
printf("AUTH: %s\n", reply->str);
freeReplyObject(reply);
/* PING server */
reply = redisCommand(c,"PING");
printf("PING: %s\n", reply->str);
freeReplyObject(reply);
/* Set a key */
reply = redisCommand(c,"SET %s %s", "foo", "hello world");
printf("SET: %s\n", reply->str);
freeReplyObject(reply);
/* Set a key using binary safe API */
reply = redisCommand(c,"SET %b %b", "bar", (size_t) 3, "hello", (size_t) 5);
printf("SET (binary API): %s\n", reply->str);
freeReplyObject(reply);
/* Try a GET and two INCR */
reply = redisCommand(c,"GET foo");
printf("GET foo: %s\n", reply->str);
freeReplyObject(reply);
reply = redisCommand(c,"INCR counter");
printf("INCR counter: %lld\n", reply->integer);
freeReplyObject(reply);
/* again ... */
reply = redisCommand(c,"INCR counter");
printf("INCR counter: %lld\n", reply->integer);
freeReplyObject(reply);
/* Create a list of numbers, from 0 to 9 */
reply = redisCommand(c,"DEL mylist");
freeReplyObject(reply);
for (j = 0; j < 10; j++) {
char buf[64];
snprintf(buf,64,"%d",j);
reply = redisCommand(c,"LPUSH mylist element-%s", buf);
freeReplyObject(reply);
}
/* Let's check what we have inside the list */
reply = redisCommand(c,"LRANGE mylist 0 -1");
if (reply->type == REDIS_REPLY_ARRAY) {
for (j = 0; j < reply->elements; j++) {
printf("%u) %s\n", j, reply->element[j]->str);
}
}
freeReplyObject(reply);
/* Disconnects and frees the context */
redisFree(c);
return 0;
```

```
}
```

Compile the preceding code.

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

Perform the test run.

```
example xxx.kvstore.aliyuncs.com 6379 instance_id password
```

So far, the C/C++ client can connect to ApsaraDB for Redis.

.net client

Procedure

Download and use the .net client.

```
git clone https://github.com/ServiceStack/ServiceStack.Redis
```

Create a .net project in the .net client.

Add the reference file stored in the library file directory ServiceStack.Redis/lib/tests to the client.

Enter the following code in the created .net project to connect to ApsaraDB for Redis. For more information about port use, see https://github.com/ServiceStack/ServiceStack.Redis.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ServiceStack.Redis;
namespace ServiceStack.Redis.Tests
{
class Program
{
public static void RedisClientTest()
{
string host = "127.0.0.1";/*IP address of the access host*/
string password = "password";/*Password*/
RedisClient redisClient = new RedisClient(host, 6379, password);
string key = "test-aliyun";
string value = "test-aliyun-value";
redisClient.Set(key, value);
string listKey = "test-aliyun-list";
System.Console.WriteLine("set key " + key + " value " + value);
string getValue = System.Text.Encoding.Default.GetString(redisClient.Get(key));
System.Console.WriteLine("get key " + getValue);
System.Console.Read();
}
public static void RedisPoolClientTest()
{
string[] testReadWriteHosts = new[] {
"redis://password@127.0.0.1:6379"/*redis://password@access address:port number*/
};
RedisConfig.VerifyMasterConnections = false;//You must set the parameter.
PooledRedisClientManager redisPoolManager = new PooledRedisClientManager(10/*Number of
connection pools*/, 10/*Connection pool timeout time*/, testReadWriteHosts);for (int i = 0; i < 100;
i++){
IRedisClient redisClient = redisPoolManager.GetClient();//Obtain the connection.
RedisNativeClient redisNativeClient = (RedisNativeClient)redisClient;
redisNativeClient.Client = null;//ApsaraDB for Redis does not support client setname. Therefore, you
must set the client object to null.
try
{
string key = "test-aliyun1111";
string value = "test-aliyun-value1111";
redisClient.Set(key, value);
string listKey = "test-aliyun-list";
redisClient.AddItemToList(listKey, value);
System.Console.WriteLine("set key " + key + " value " + value);
string getValue = redisClient.GetValue(key);
System.Console.WriteLine("get key " + getValue);
redisClient.Dispose();//
}catch (Exception e)
{
System.Console.WriteLine(e.Message);
}
}
System.Console.Read();
}static void Main(string[] args)
{
//Single connection mode
RedisClientTest();
//Connection pool mode
RedisPoolClientTest();
}
}
}
```

node-redis client

Procedure

Download and install node-redis.

```
npm install hiredis redis
```

Enter and run the following code in the node-redis client to connect to ApsaraDB for Redis.

```
 var redis = require("redis"),
client = redis.createClient({detect_buffers: true});
client.auth("password", redis.print)
```

Use ApsaraDB for Redis.

```
   // Write data.
client.set("key", "OK");
// Obtain data and a string is returned.
client.get("key", function (err, reply) {
console.log(reply.toString()); // print `OK`
});
// If a buffer is transmitted, a buffer is returned.
client.get(new Buffer("key"), function (err, reply) {
console.log(reply.toString()); // print `<Buffer 4f 4b>`
});
client.quit();
```

ApsaraDB for Redis only supports access from Alibaba Cloud intranet. It does not support Internet accesses. That is, only clients of ApsaraDB for Redis installed on ECSs of the same node can be connected to ApsaraDB for Redis for data operations.

> **Note**: Redis-cli is the native command line interface for Redis. You can first download and install Redis on ECS before using Redis-cli. For the Redis installation commands, refer to the official doucment here.

You can run the following redis-cli command to connect to ApsaraDB for Redis:

```
redis-cli -h instance connection address -a Password
```

Prerequisites

To access an ApsaraDB for Redis instance from a local PC to operate data, configure the port mapping or forwarding on ECS. However, the following prerequisites must be met:

If the ApsaraDB for Redis instance is in a VPC, ECS and the ApsaraDB for Redis instance must be in the same VPC.

If the ApsaraDB for Redis instance is in a classic network, ECS and the ApsaraDB for Redis instance must be in the same node (region).

If an IP address whitelist is configured for the ApsaraDB for Redis instance, the ECS Intranet address must be added to the whitelist.

ECS Windows

Currently, ApsaraDB for Redis is accessible through ECS Intranet. To locally access ApsaraDB for Redis through a public network, perform port mapping using netsh on the ECS Windows server.

Log on to the ECS Windows server and run the following command in CMD:

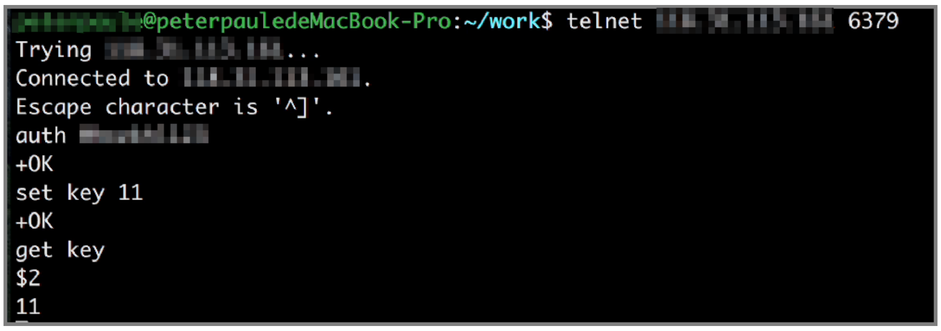
```
 netsh interface portproxy add v4tov4 listenaddress=private IP address of the ECS server
listenport=6379 connectaddress=connection address of ApsaraDB for Redis connectport=6379
```

Where,

netsh interface portproxy delete v4tov4 listenaddress=public IP address of the ECS server listenport=6379 can be used to delete unnecessary mappings.

netsh interface portproxy show all can be used to show existing mappings on the current server.

Perform a verification test locally after configuration is complete.



```
[redacted]@peterpauledeMacBook-Pro:~/work$ telnet [redacted] 6379
Trying [redacted]...
Connected to [redacted].
Escape character is '^]'.
auth [redacted]
+OK
set key 11
+OK
get key
$2
11
```

Run redis-cli locally to connect to the ECS Windows server. For example, if the IP address of the ECS Windows server is 1.1.1.1, you can telnet to 1.1.1.1 6379.

After the ECS Windows server is connected, enter the password to connect to ApsaraDB for Redis: auth Redis connection password.

Write data and perform query and verification.

After performing the preceding steps, you can use a local PC or server to connect to port 6379 of the ECS Windows server through a public network and access ApsaraDB for Redis.

Note: As portproxy is provided by Microsoft rather than open source software, read the netsh documentation on portproxy or consult Microsoft engineers if you have any problems in the configuration or usage process. Alternatively, use another scheme, for example, use portmap to configure proxy mappings.

ECS Linux

Currently, ApsaraDB for Redis is accessible through ECS Intranet. To locally access ApsaraDB for Redis through a public network, install rinetd on the ECS Linux server to perform forwarding.

Install rinetd on the ECS Linux server.

```
 wget http://www.boutell.com/rinetd/http/rinetd.tar.gz&&tar -xvf rinetd.tar.gz&&cd rinetd
sed -i 's/65536/65535/g' rinetd.c (Modify the port range.)
mkdir /usr/man&&make&&make install
```

Note: The rinetd installation package obtained from the download URL may be unavailable. You can find and download the rinetd installation package from other sources.

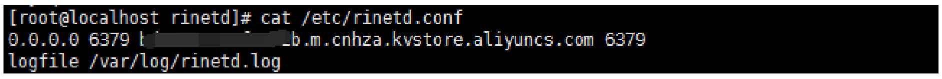
Open the configuration file rinetd.conf.

```
vi /etc/rinetd.conf
```

Add the following content to the configuration file:

```
 0.0.0.0 6379 Connection address of port 6379 of ApsaraDB for Redis
logfile /var/log/rinetd.log
```

Note: You can run cat /etc/rinetd.conf to check whether the configuration file is correctly modified.



```
[root@localhost rinetd]# cat /etc/rinetd.conf
0.0.0.0 6379 [redacted]b.m.cnhza.kvstore.aliyuncs.com 6379
logfile /var/log/rinetd.log
```

Run the following command to start rinetd.

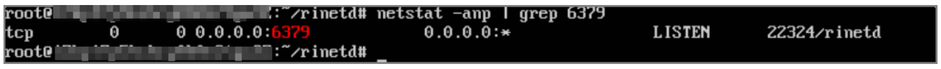
```
rinetd
```

Notes:

You can run echo rinetd >>/etc/rc.local to set auto startup for rinetd.

If a binding error is reported, run pkill rinetd to terminate the process and run rinetd to start the rinetd process.

After rinetd is started normally, run netstat -anp | grep 6379 to check whether the service works properly.



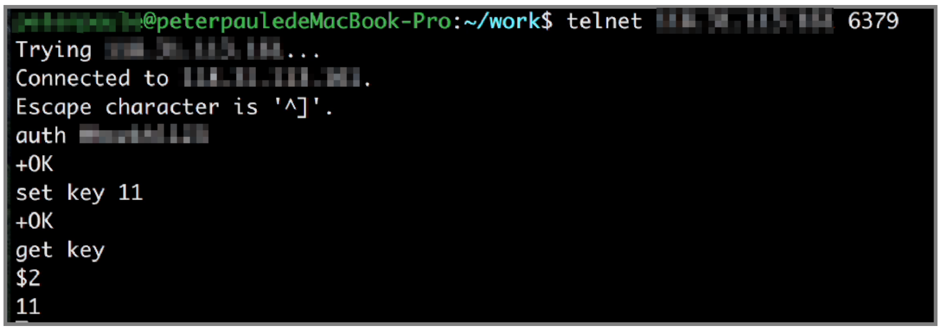
```
root@[redacted]:~/rinetd# netstat -anp | grep 6379
tcp        0      0 0.0.0.0:6379            0.0.0.0:*               LISTEN      22324/rinetd
root@[redacted]:~/rinetd# _
```

Perform a verification test locally.

You can run redis-cli locally to connect to the ECS Linux server for logon verification. For example, if the IP address of the server with rinetd installed is 1.1.1.1, run redis-cli -h 1.1.1.1 -a ApsaraDB for Redis instance ID:ApsaraDB for Redis password. Alternatively, telnet to the ECS Linux server and perform the operation verification. For example, if the IP address of the ECS Linux server is 1.1.1.1, you can telnet to 1.1.1.1 6379.

After the ECS Linux server is connected, enter the password to connect to ApsaraDB for Redis: auth Redis connection password.

Write data and perform query and verification.



```
[redacted]@peterpauledeMacBook-Pro:~/work$ telnet [redacted] 6379
Trying [redacted]...
Connected to [redacted].
Escape character is '^]'.
auth [redacted]
+OK
set key 11
+OK
get key
$2
11
```

After performing the preceding steps, you can use a local PC or server to connect to port 6379 of the ECS Linux server through a public network and access ApsaraDB for Redis.

Note: You can use the preceding scheme to test and use rinetd. As rinetd is open source software, read its official documentation or contact rinetd engineers for help if you have any problems in use.