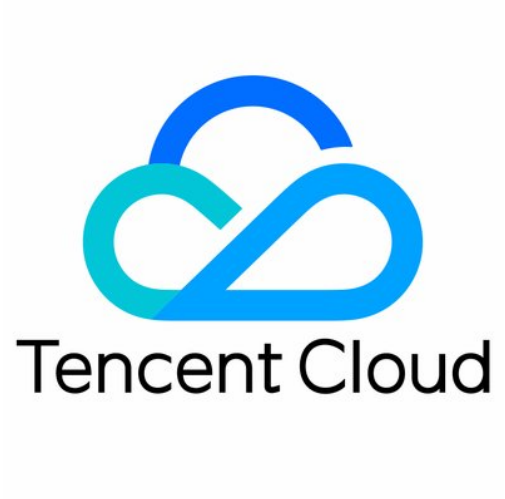
Cloud Load Balancer

Ops Guide

Product Documentation



Contents

Ops Guide
Solution to Excessive Clients in TIME_WAIT Status

Last updated：2024-01-04 14:39:00

Background

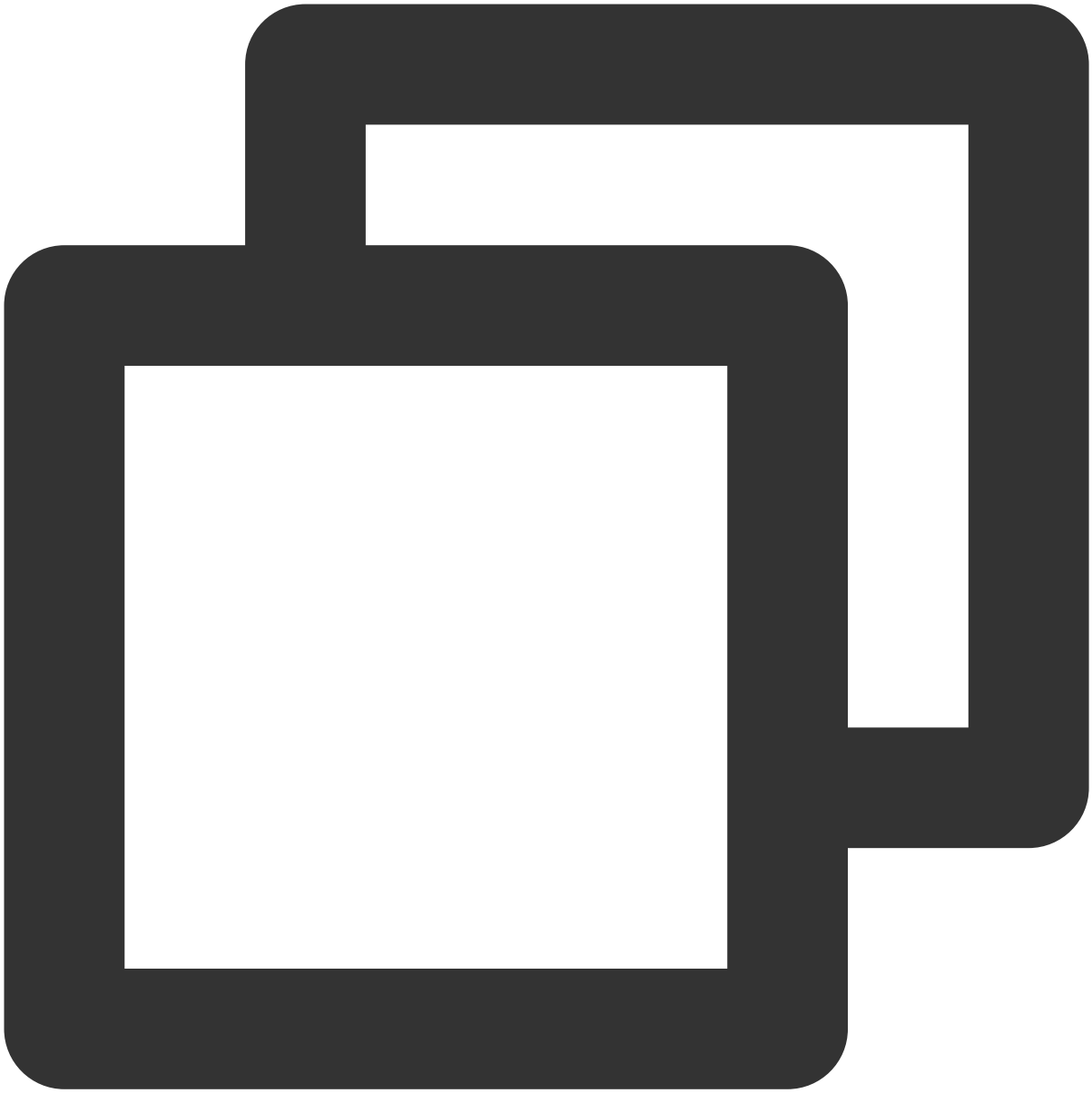
When performing stress testing on CLB, you may encounter connection failures caused by too many client TIME-WAIT (all ports are occupied in short time). Below are reasons and solutions:

Linux parameter description

tcp_timestamps : Whether to enable TCP timestamps. A timestamp is negotiated in TCP three-way handshake. If either party does not support this parameter, it will not be used in this connection.
tcp_tw_recycle : Whether to enable reuse of TCP TIME-WAIT state.
tcp_tw_reuse : When enabled, connections in TIME_WAIT state that exceeds 1 second can be directly reused.

Cause Analysis

The client has too many TIME_WAIT because it proactively closes connections. When the client closes a connection, the connection will enter TIME_WAIT state and be reused after 60 seconds by default. In this case, you can enable `tcp_tw_recycle` and `tcp_tw_reuse` parameters to facilitate reuse of connections in TIME_WAIT state. If `tcp_timestamps` is currently disabled on CLB, `tcp_tw_recycle` and `tcp_tw_reuse` parameters enabled by the client will not take effect, and connections in TIME_WAIT state cannot be quickly reused. The following describes some Linux parameters and reasons why `tcp_timestamps` cannot be enabled on CLB:
1. `tcp_tw_recycle` and `tcp_tw_resuse` only take effect when `tcp_timestamps` is enabled.
2. In a `FULLNAT` scenario, `tcp_timestamps` and `tcp_tw_recycle` cannot be enabled at the same time, because the public network client may fail to access the server through the NAT gateway. The reasons are as follows: If both tcp_tw_recycle and tcp_timestamps are enabled, timestamp in the socket connect requests of the same source IP (same server) must be incremental within 60 seconds. Taking 2.6.32 kernel as an example, the details are as follows:



```
if(tmp_opt.saw_tstamp && tcp_death_row.sysctl_tw_recycle &&
(dst = inet_csk_route_req(sk,req))!= NULL &&
 (peer = rt_get_peer((struct rtable *)dst))!= NULL &&
 peer->v4daddr == saddr){
 if(get_seconds()< peer->tcp_ts_stamp + TCP_PAWS_MSL &&
     (s32)(peer->tcp_ts - req->ts_recent) > TCP_PAWS_WINDOW){
         NET_INC_STATS_BH(sock_net(sk),LINUX_MIB_PAWSPASSIVEREJECTED) ;
         goto ↓drop_and_release ;
 }
}
```

Note:

tmp_opt.saw_tstamp: This socket supports `tcp_timestamp` .

sysctl_tw_recycle: `tcp_tw_recycle` has been enabled for this server.

TCP_PAWS_MSL: 60s; the last TCP communication of the source IP occurred within 60 seconds.

TCP_PAWS_WINDOW: 1; the `timestamp` of the last TCP communication of the source IP is greater than that of this TCP communication.

3. On CLB (Layer-7), `tcp_timestamps` is disabled because the public network client may fail to access the server through the NAT gateway, as shown in the example below:

a. A quintuple is still in TIME_WAIT state. In the port allocation policy of the NAT gateway, the same quintuple is reused in twice the maximum segment lifetime (2MSL), and a SYN packet is sent.

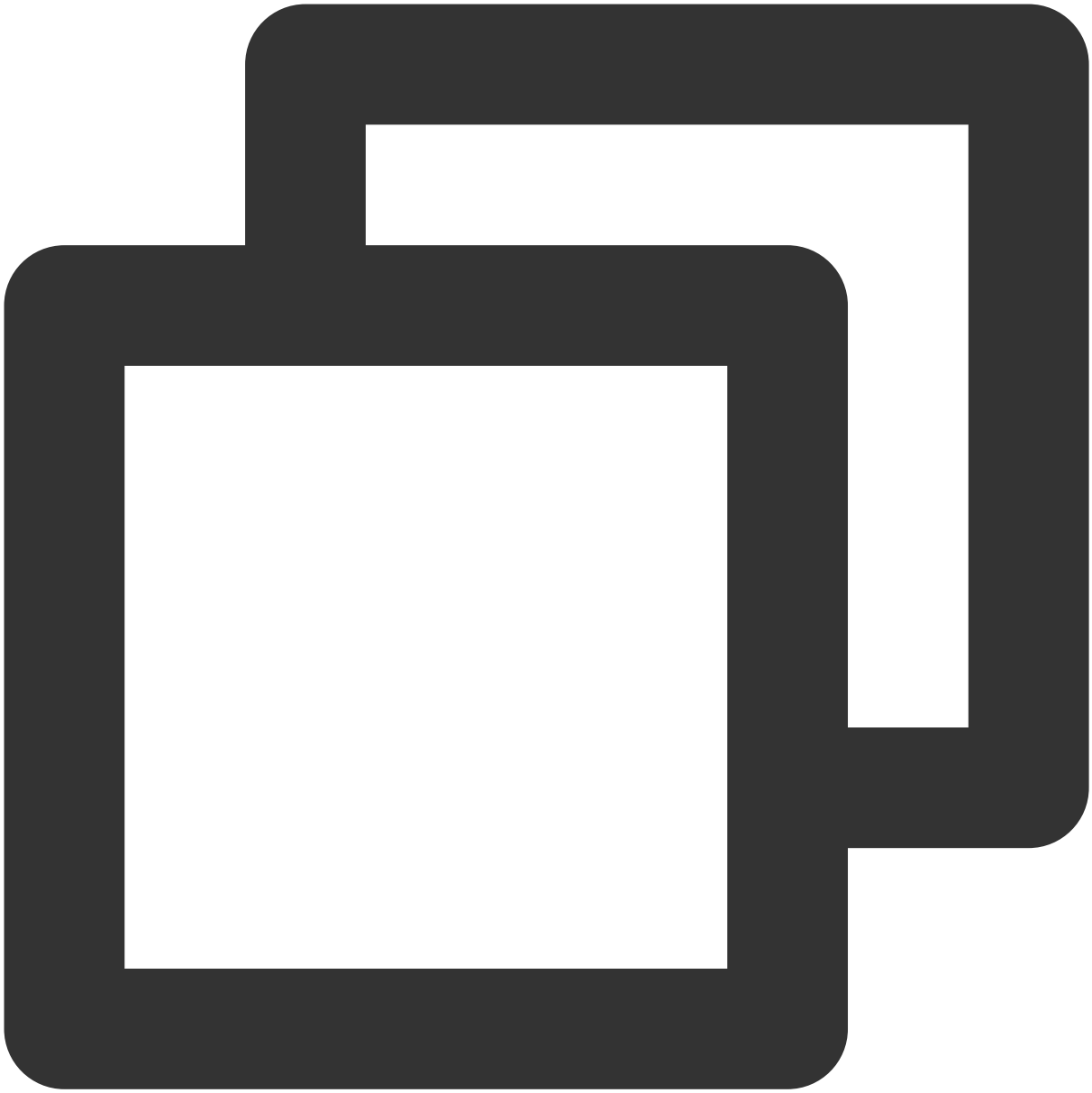
b. When `tcp_timestamps` is enabled and the following two conditions are met, the SYN packet will be dropped (because timestamp option is enabled, and the packet is considered as old).

i. Timestamp of last time > Timestamp of this time

ii. Packets are received within 24 days (the timestamp field is 32-bit and the timestamp is updated once per 1 millisecond by default in Linux. The timestamp will wrap around after 24 days).

Note:

This problem is more obvious on mobile devices because clients share limited public network IPs under the ISP's NAT gateway and a quintuple can be reused in 2MSL. The timestamps sent from different clients may not be incremental. Taking 2.6.32 kernel as an example, the details are as follows:



```
static inline int tcp_paws_check(const struct tcp_options_received *rx_opt,int paws
{
 if((s32)(rx_opt->ts_recent - rx_opt->rcv_tsval)<= paws_win)
 return 1;
 if(unlikely(get_seconds()>=rx_opt->ts_recent_stamp + TCP_PAWS_24DAYS))
 return 1;
 return 0;
}
```

Note:

rx_opt->ts_recent: Timestamp of last time
rx_opt->rcv_tsval: Timestamp received in this time
get_seconds(): Current time
rx_opt->ts_recent_stamp: Time when the previous packet was received

Solution

If the client has too many TIME_WAIT, see below for solutions:
HTTP uses non-persistent connections (Connection: close). In this case, CLB proactively closes the connection, and the client will not generate TIME_WAIT.
If the scenario requires a persistent connection, enable SO_LINGER option of the socket and use RST to close the connection to avoid the TIME_WAIT state and achieve fast port reuse.

Load Balancer HTTPS Service Performance Test

Last updated：2024-01-04 14:39:00

1. HTTPS capability of CLB

By optimizing the protocol stack and server, Tencent Cloud CLB significantly improves the HTTPS performance. Meanwhile, Tencent Cloud substantially reduces the cost of certificates via international cooperation. Tencent CLB can benefit your business in the following aspects:
1. The use of HTTPS does not affect the access speed of the client.
2. SSL encryption and decryption performance of a single server in a cluster supports full handshakes of up to 65,000 connections per second (CPS), which is at least 3.5 times higher than that of a high-performance CPU. This reduces server cost, significantly improves service capability during business operation and traffic spikes, and strengthens the anti-attack capability.
3. Offloading and conversion of multiple protocols are supported, which reduce the stress of business adapting to various client protocols. The business backend only needs to support HTTP/1.1 to use different versions of protocols, such as HTTP/2, SPDY, SSL 3.0 and TLS 1.2.
4. One-stop SSL certificate application, monitoring, and replacement are supported. Tencent Cloud cooperates with international certificate authorities, Comodo and Symantec, to simplify the certificate application process and reduce cost.
5. Anti-CC and WAF features are provided to effectively eliminate application-layer attacks, such as slow HTTP attacks, high-frequency targeted attacks, SQL injection, and website trojans.

2. Test Purpose

HTTPS service has advantages such as identity authentication, information encryption, and integrity verification. However, using SSL protocol to implement secure communication results in certain performance loss, including increased latency and CPU resource consumption by encryption and decryption. This document includes extreme performance test data of Tencent Cloud's HTTPS services during SSL encryption and decryption. You can compare it with the traditional HTTPS performance data.

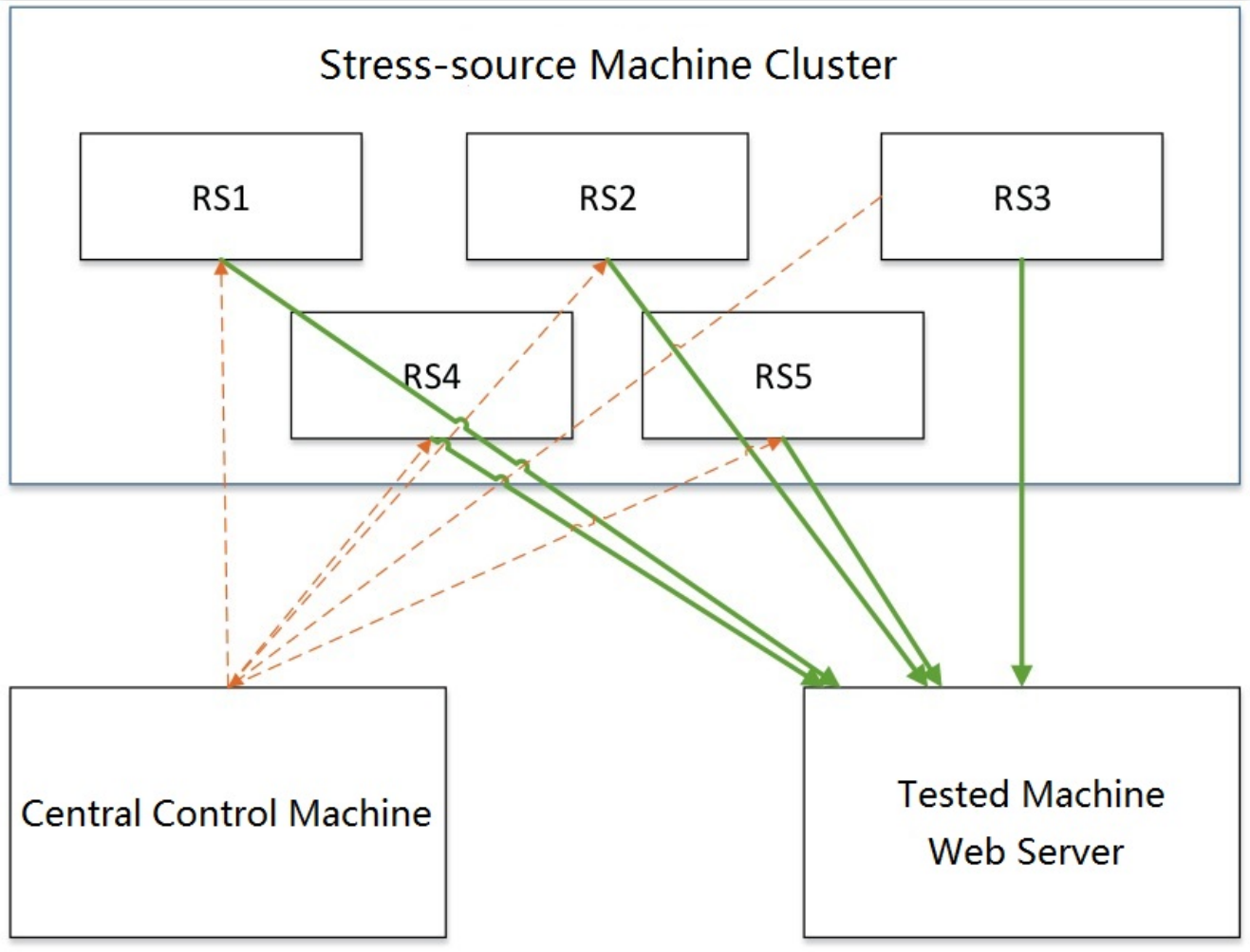
3. Test Environment

Stress testing tool: wrk 4.0.2

Tencent Cloud's underlying service environment: Nginx 1.1.6–1.9.9 + OpenSSL 1.0.2h
Information on the OS where Nginx is installed: Linux TENCENT64.site 3.10.94-1-tlinux2-0036.tl2 # 1 SMP Thu Jan 21 03:40:59 CST 2016 x86_64 x86_64 x86_64 GNU/Linux
OS of other stress servers: Linux TENCENT64.site 2.6.32.43-tlinux-1.0.17-default #1 SMP Tue Nov 17 18:03:12 CST 2015 x86_64 x86_64 x86_64 GNU/Linux

4. WebServer cluster test scheme

The stress from a single server is not enough to test the performance limit of Tencent Cloud's HTTPS service. Multiple stress servers are needed. The test includes three parts:
1. Stress testing cluster. It is used to distribute HTTP/HTTPS stress and output the stress testing result of a single stress server.
2. Central control server, which synchronously controls the start and end of the stress testing cluster, obtains testing data from each stress server, aggregates and outputs the data.
3. Web server, which is the CVM instance hosting Tencent Cloud's HTTPS service. When WebServer performance is tested, a page can be returned directly without upstream connection.
The connection is as follows:



5. Performance test data of HTTPS WebServer

Connection Type	Session Cache	Packet Size (in bytes)	Encryption Suite	Performance (QPS)
Persistent	On	230	ECDHE-RSA-AES128-GCM-SHA256	296241
Non-Persistent	Off	230	ECDHE-RSA-AES128-GCM-SHA256	65630

6. CLB HTTPS capability test conclusion

According to the table above, Tencent Cloud's HTTPS service supports SSL encryption and decryption. It has multiple server clusters on the backend, and a single server in a cluster can achieve a performance of up to 65,000

QPS during full handshake and of about 300,000 QPS during persistent connection.
In normal circumstances, HTTPS protocol adds at least one full handshake process when using SSL protocol, and latency increases by 2 * round-trip time (RTT). In addition, SSL symmetric/asymmetric encryption consumes a large amount of CPU resources. The decryption capability of RSA is a major hurdle for HTTPS-based access.
With Tencent Cloud's HTTPS service, you do not need to deploy additional services for SSL encryption and decryption. With no additional fee charged, the service allows you to enjoy powerful business-hosting and anti-attack capabilities.

Stress Testing FAQ

Last updated：2024-01-04 14:39:00

Based on customer experiences in stress testing, this document summarizes common performance issues in stress testing, and provides troubleshooting solutions as well as suggestions.

Stress testing FAQs

The public network access is not enabled on real server

If public network access is not enabled when you purchase CVM, forwarding may fail when a public network CLB is mounted to the CVM instance.

The bandwidth of a real server is insufficient

If the real server has a low bandwidth, it cannot return packets to the CLB when the threshold is exceeded. CLB will return a 504 or 502 error to the client.

The client ports are insufficient

If the number of clients is too small or the range of client ports is too narrow, client ports become insufficient and connections will fail to be established. In addition, if the `keep_alive` value is greater than 0 when a persistent connection is established, the connection will permanently use the port, which reduces the number of available client ports.

Applications relied on by real servers have performance issues

After a request reaches a real server through CLB, the load on the real server is normal. However, because applications on real servers also rely on other applications such as database, performance issues in the database may also affect the stress testing performance.

A real server is unhealthy

The health status of real servers may be ignored in stress testing. If the real server has a health check failure or unstable health check status (sometimes good and sometimes bad with rapid changes), stress testing may have poor performance.

Session persistence enabled for CLB results in uneven traffic distribution among real servers

After session persistence is enabled for CLB, requests may be distributed to fixed real servers. The traffic distribution becomes uneven, affecting the performance of stress testing. We recommend you disable session persistence during

stress testing.

Suggestions for stress testing

Note:

The following configurations are only used for CLB stress testing. You do not need to have them in your production environment.

We recommend you use non-persistent connection when stress testing the forwarding capability of CLB.

Except for verification on session persistence features, stress testing is generally designed to verify the forwarding capability of CLB. Therefore, non-persistent connection can be used to test the processing capability of CLB and real servers.

We recommend you use persistent connection to stress testing the throughput of CLB, such as the upper limit of bandwidth and persistent connection services.

We recommend you adjust the timeout period of the stress testing tool to a small value. Otherwise, the average response time will increase when the timeout period increases, causing you to be unable to quickly judge whether the stress level is reached.

We recommend you use a static website provided by the real server for stress testing to avoid loss caused by application logic, such as I/O and DB.

Disable session persistence for listeners. Otherwise, the stress will be concentrated on certain real servers. If the pressure performance is not satisfactory, you can determine whether the traffic is evenly distributed by checking the monitoring data of real servers under CLB.

Disable health check for listeners to reduce access requests to real servers generated during health check.

Use multiple clients (> 5) for stress testing. Dispersed source IPs can better simulate actual online conditions.

CLB Certificate Operation Permissions

Last updated：2024-01-04 14:39:00

Operation Scenarios

Since March 23, 2020, all certificate operations of CLB have been connected to Cloud Access Management (CAM) for authentication. Therefore, when a sub-user account performs CLB certificate operations, if "You are not authorized for this operation. Please contact your developer." is displayed, you can grant certificate permissions to the sub-user account as instructed below.

Prerequisites

The logged-in account needs to be the root account or a sub-user account with CAM permissions (i.e., associated with the `QcloudCamFullAccess` policy).

Note:

To check whether the sub-user account has CAM permissions, go to User List in the CAM Console, enter the details page of the sub-user, and check whether the `QcloudCamFullAccess` policy has been associated.

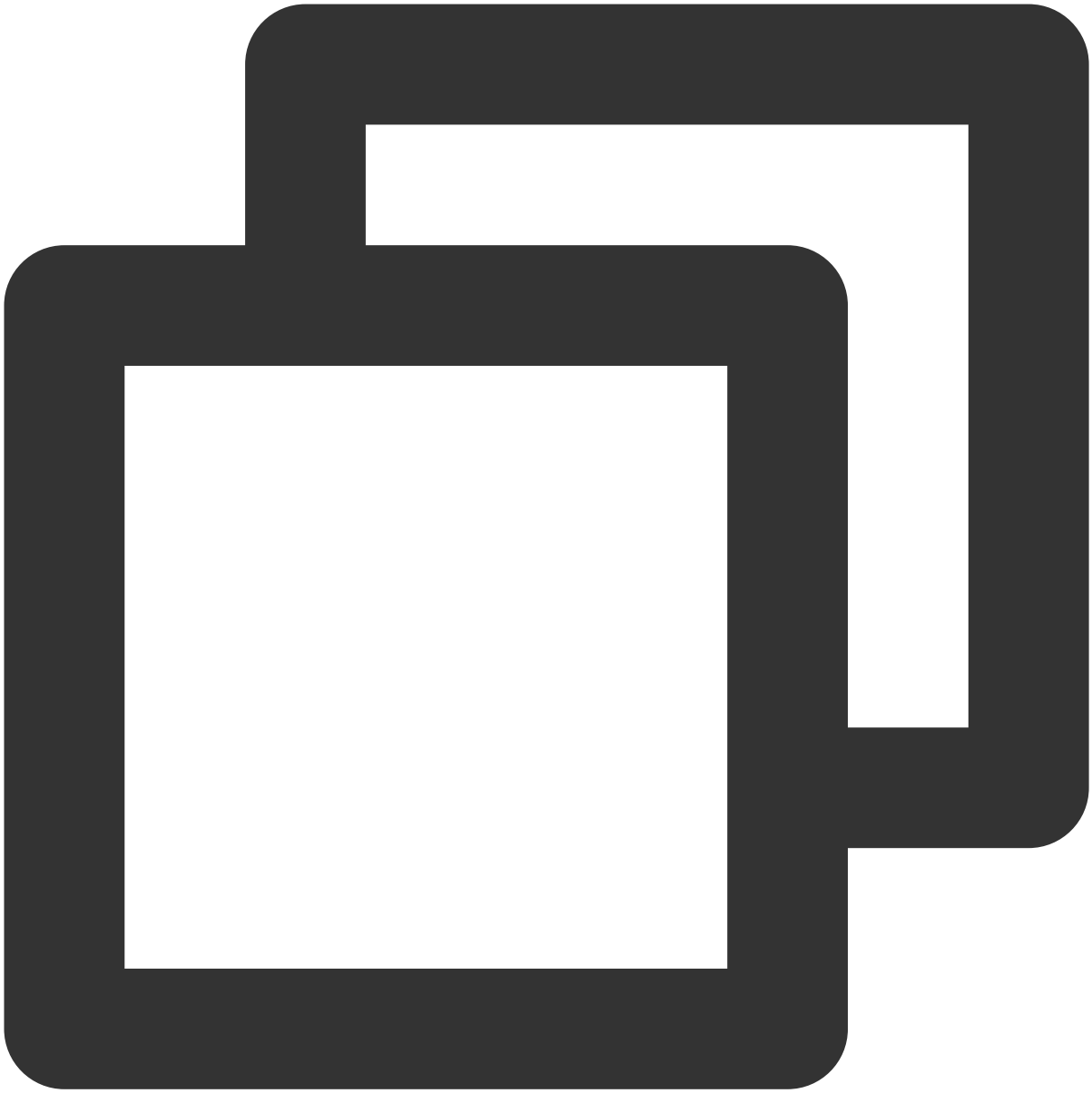
If the `QcloudCamFullAccess` policy is associated, but "No API permissions (message:GetReceiversOnAllType). Please contact your developer." is displayed when the sub-user performs certificate operations, they can ignore and proceed anyway.

Directions

Please grant certificate permissions in the following methods:

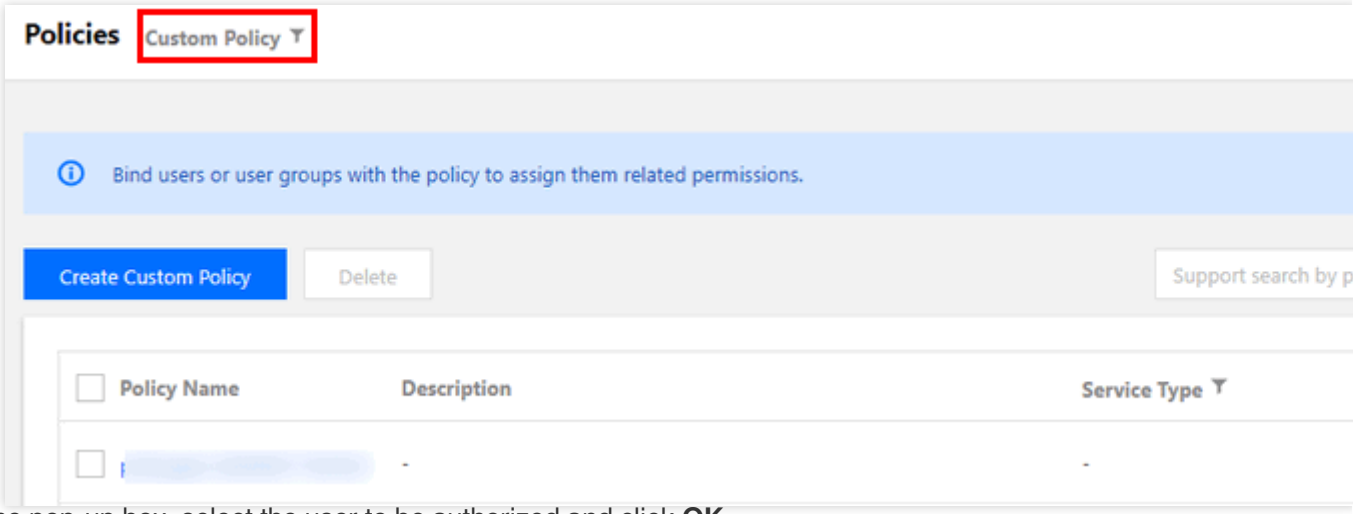
Method 1. Associate a custom policy

1. Log in to the CAM Console.
2. On the left sidebar, click **Policies**.
3. Click **Create Custom Policy** and select **Create by Policy Syntax** in the pop-up box.
4. On the "Select Template Policy" page, select **Blank Template** and click **Next**.
5. On the "Edit Policy" page, enter the policy name and enter the following policy content in the "Edit Policy Content" input box:

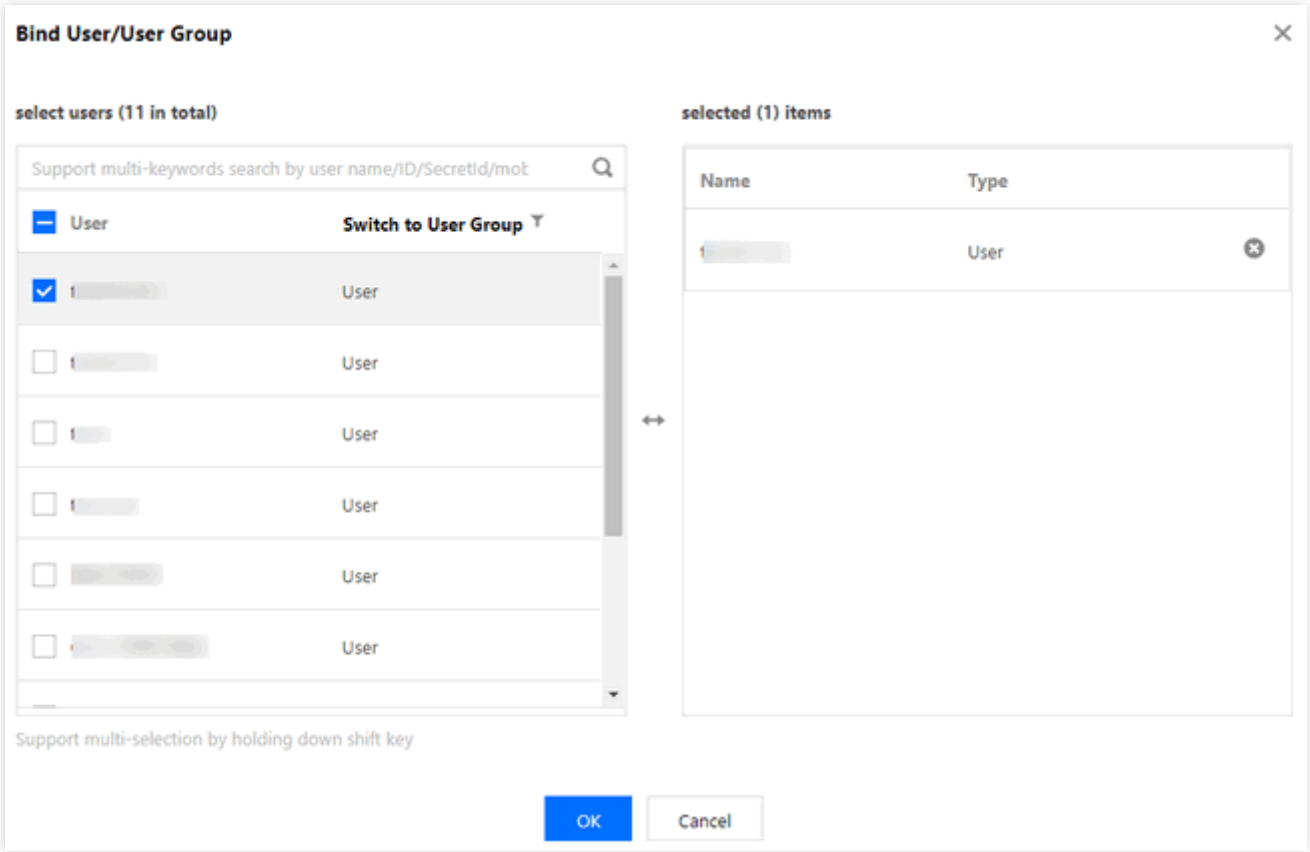


```
{
 "version": "2.0",
 "statement": [
     {
         "action": "name/ssl:*",
         "resource": "qcs::ssl:::*",
         "effect": "allow"
     }
 ]
}
```

6. Then, click **Done** to return to the "Policy" list page.
7. At the top of the "Policy" list page, select **Custom Policy**, find the row of the policy you just created in the list, and click **Associate User/Group** in the "Operation" column.



8. In the pop-up box, select the user to be authorized and click **OK**.



Method 2. Associate a preset policy

1. Log in to the CAM Console.
2. On the left sidebar, select **User** > **User List** to enter the "User List" page.
3. In the row of the sub-user to be authorized, click **Authorize** in the "Operation" bar.

4. In the pop-up box, select `QcloudSSLFullAccess` or `QcloudSSLReadOnlyAccess` and click **OK**.