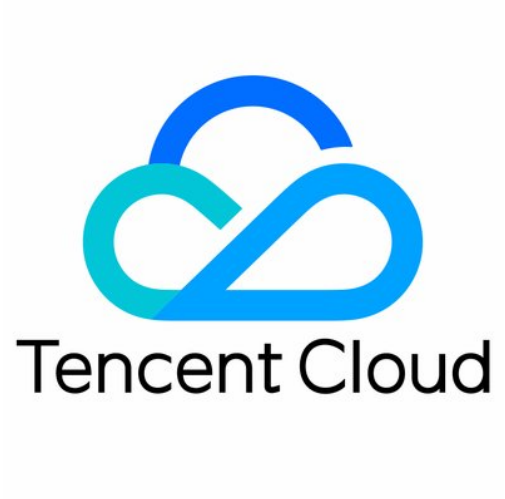
Cloud Load Balancer

Best Practices

Product Documentation



Contents

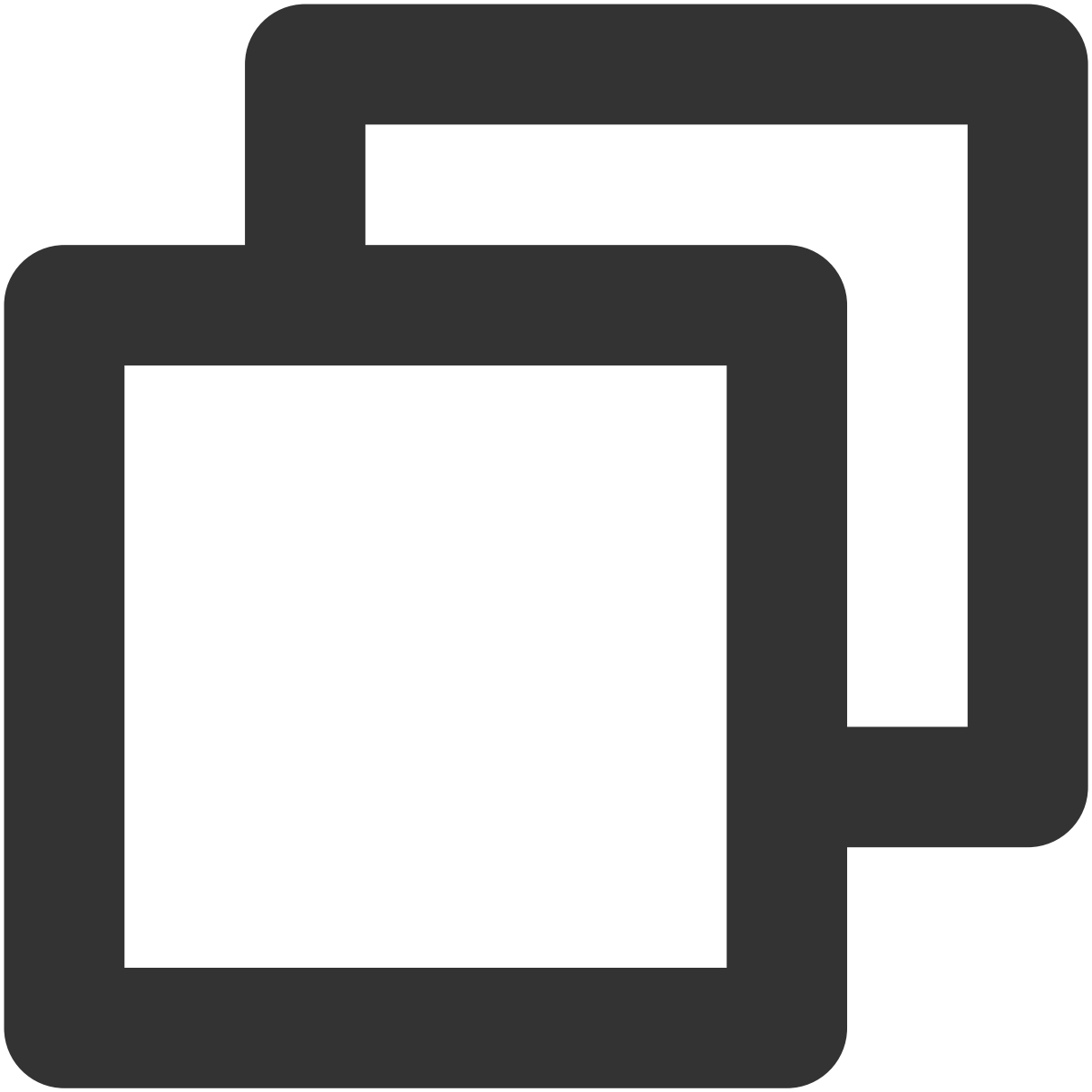
Best Practices
Enabling Gzip Compression & Testing

Last updated：2024-01-04 14:39:00

For **public network CLB instances or CLB instances with a static public IP address**, Gzip compression is enabled for the **HTTP and HTTPS** protocols by default. Gzip compresses website files before they are sent to the client browser. This effectively reduces the data volume in network transmission and speeds up webpage loading on the client browser. When using this feature, pay attention to the following:

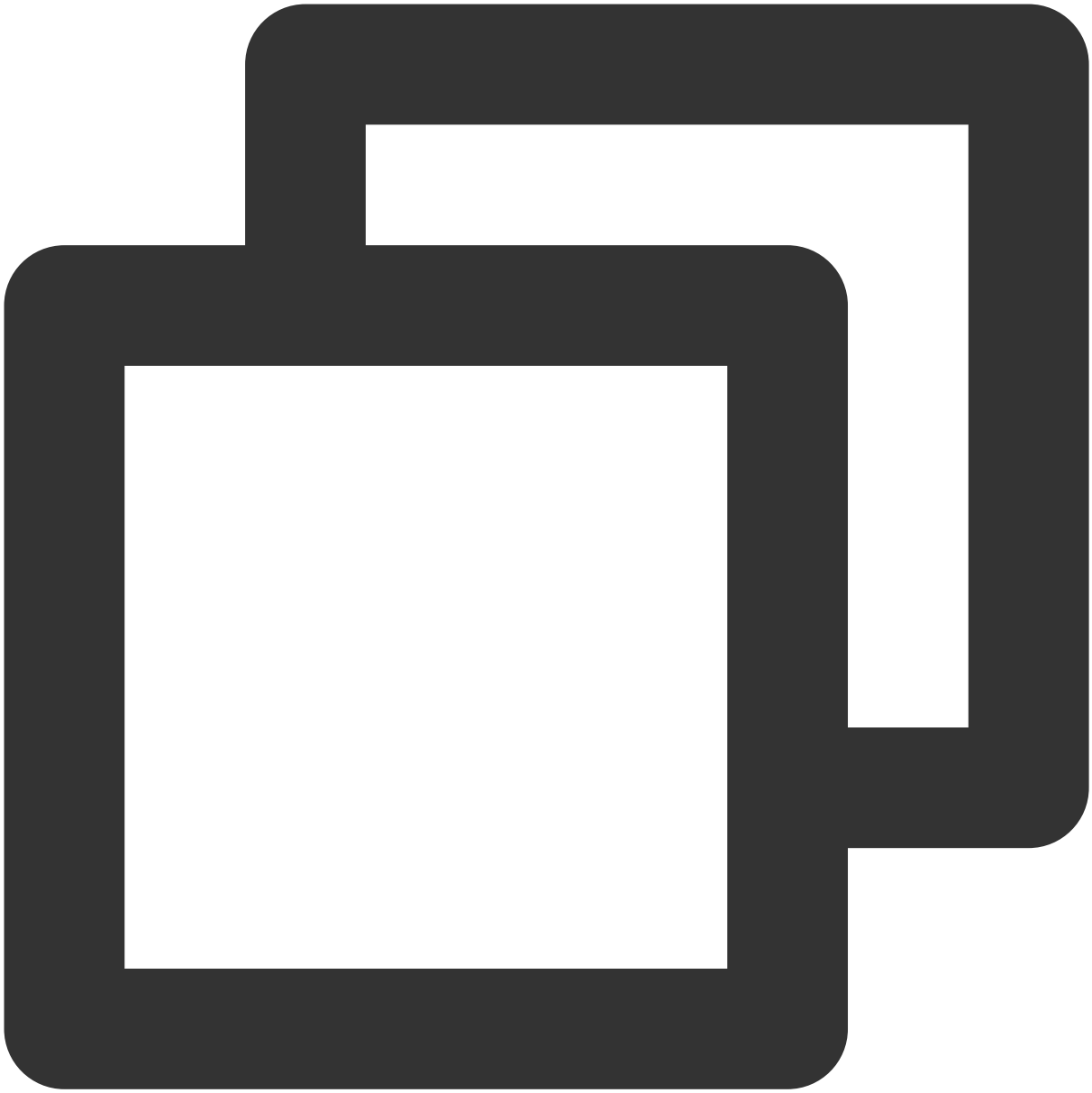
Must-knows

You must also enable Gzip compression on the backend CVM instances of CLB.
For common Nginx service containers, you must enable Gzip compression in their configuration files (nginx.conf by default) and restart the service.



```
gzip on;
```

Currently, CLB supports the following file types. You can use `gzip_types` to specify the file types for compression.



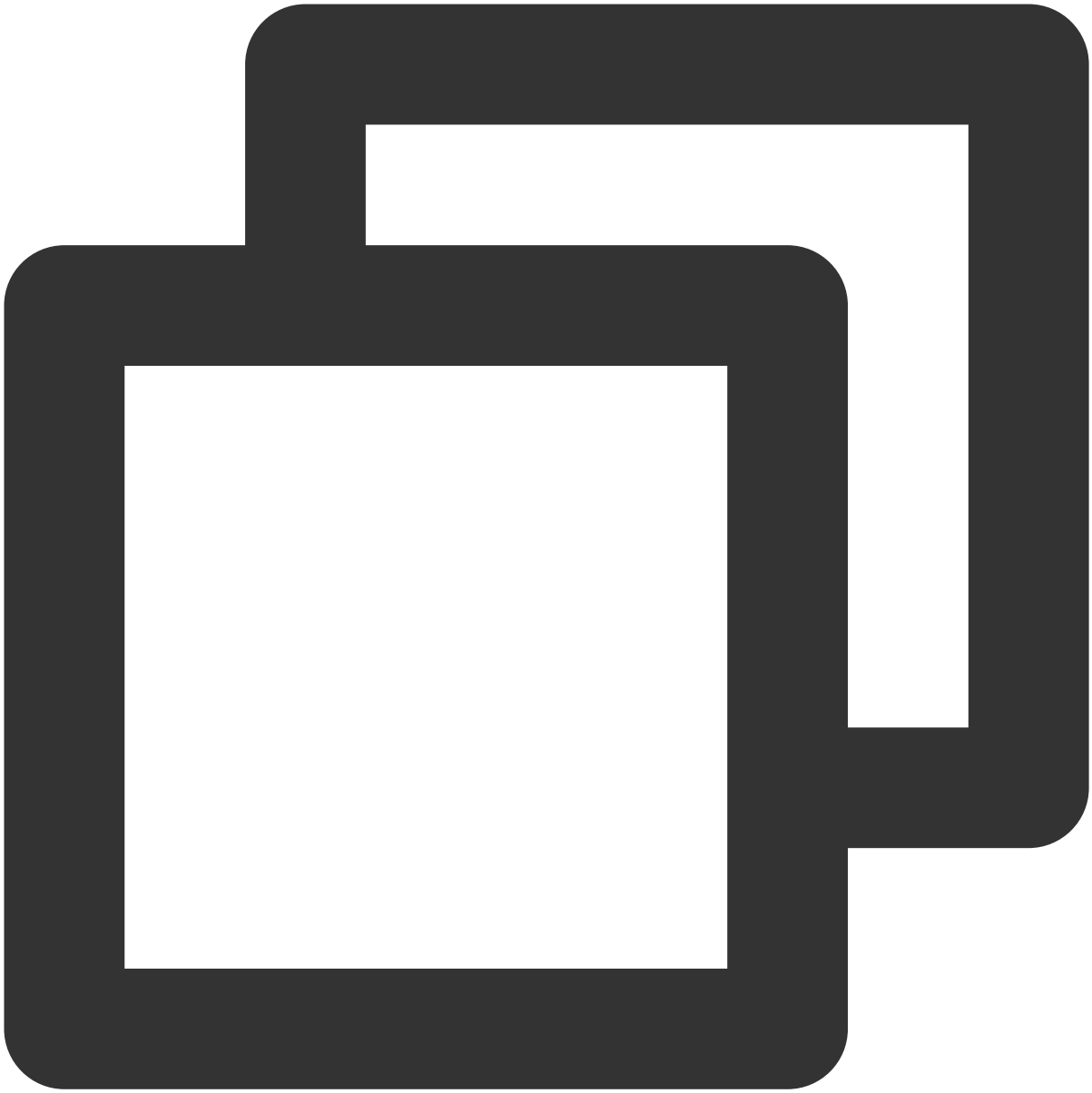
```
application/atom+xml application/javascript application/json application/rss+xml ap
```

Note:

You must enable Gzip compression for the above file types in the business software of the backend CVM instances of CLB.

The client requests must carry the compression request identifier.

To enable Gzip compression, the client requests must carry the following identifier:

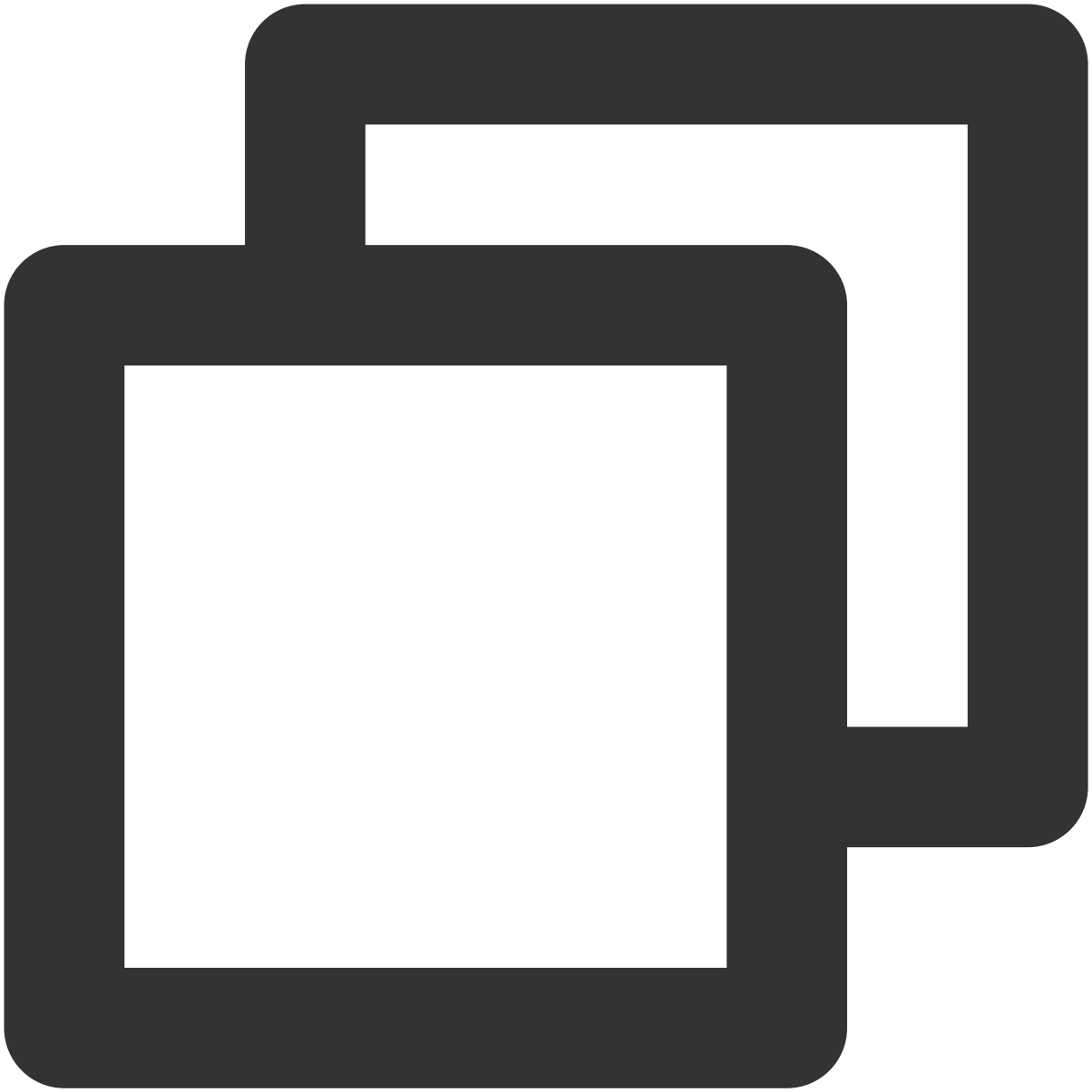


```
Accept-Encoding: gzip,deflate,sdch
```

Example of enabling Gzip compression on CVM instances

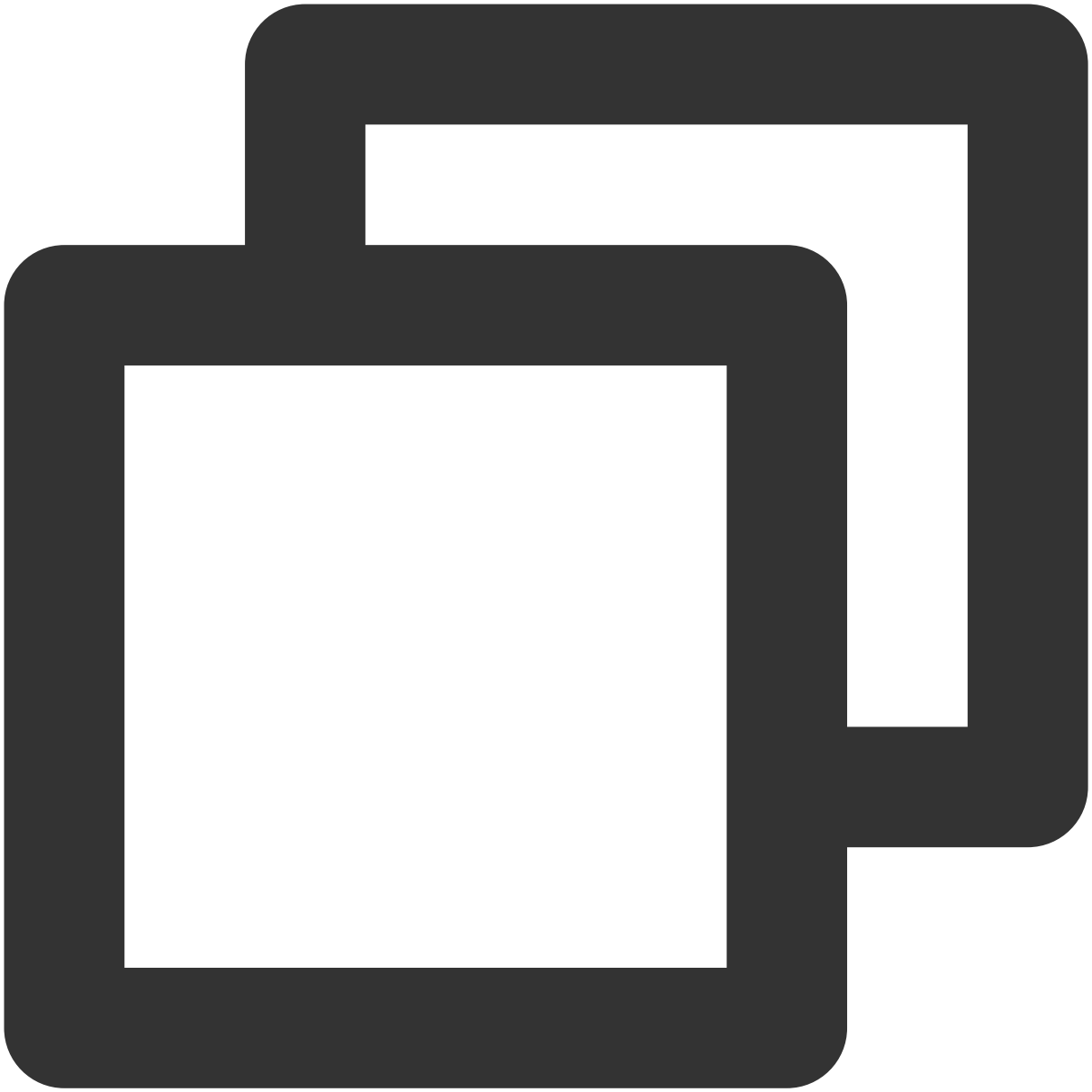
Example of CVM runtime environment: Debian 6

1. Use Vim to open the Nginx configuration file in the user path:



```
vim /etc/nginx/nginx.conf
```

2. Find the following code:



```
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.1;
gzip_comp_level 2;
gzip_types text/html application/json;
```

Description of the above code syntax:

gzip: specifies whether to enable or disable the Gzip module.

Syntax: `gzip on/off`

Scopes: http, server, location

gzip_min_length: specifies the minimum number of bytes that a page can be compressed to. The number of bytes can be obtained from `Content-Length` in the HTTP header. The default value is 1k.

Syntax: `gzip_min_length length`

Scopes: http, server, location

gzip_buffers: specifies the unit of buffer for storing the data stream of the Gzip compression result. 16k means 16k is used as the unit, and a size of memory that is 4 times the original data size (in 16k) will be applied for.

Syntax: `gzip_buffers number size`

Scopes: http, server, location

gzip_http_version: specifies the lowest HTTP version that can use Gzip. Value HTTP/1.0 means the lowest HTTP version that needs Gzip is 1.0, so Gzip can be compatible with HTTP/1.1 or higher. You do not need to change the parameter value since Tencent Cloud supports HTTP/1.1 across the entire network.

Syntax: `gzip_http_version 1.0 | 1.1;`

Scopes: http, server, location

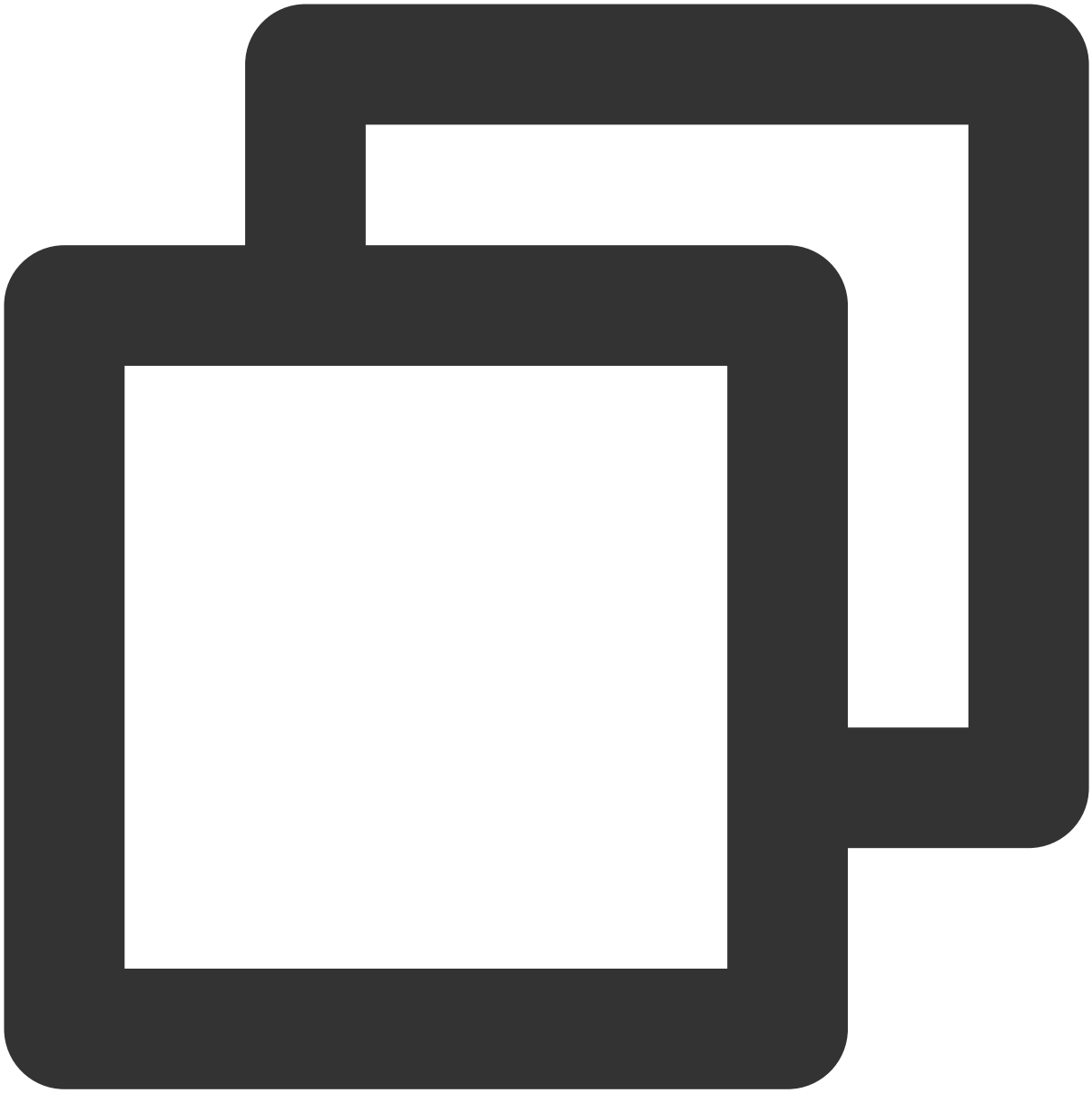
gzip_comp_level: specifies the Gzip compression ratio with a value range of 1–9. Value 1 is the smallest compression ratio with the fastest processing speed, while value 9 is the greatest compression ratio with the slowest processing speed (fast transmission with high CPU consumption).

Syntax: `gzip_comp_level 1..9`

Scopes: http, server, location

gzip_types: specifies the Multipurpose Internet Mail Extensions (MIME) types for compression. The "text/html" type will be compressed by default. In addition, Gzip for Nginx does not compress static resource files such as JavaScript files and images by default. You can configure gzip_types to specify MIME types to be compressed. Types that are not specified will not be compressed. **For example, to compress data in JSON format, you need to add application/json to the parameter value**.

The supported types are below:

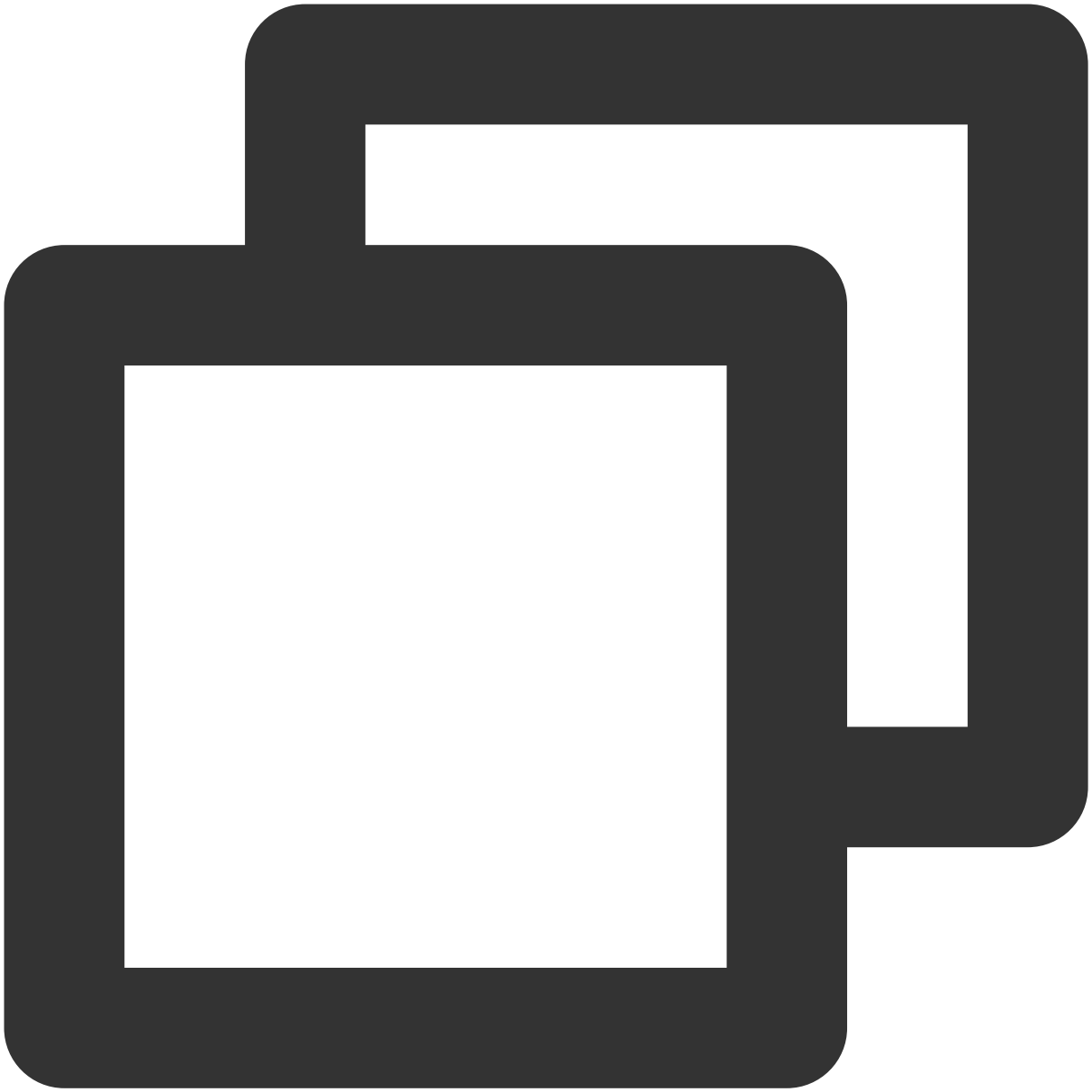


```
text/html text/plain text/css application/x-javascript text/javascript application/
```

Syntax: `gzip_types mime-type [mime-type ...]`

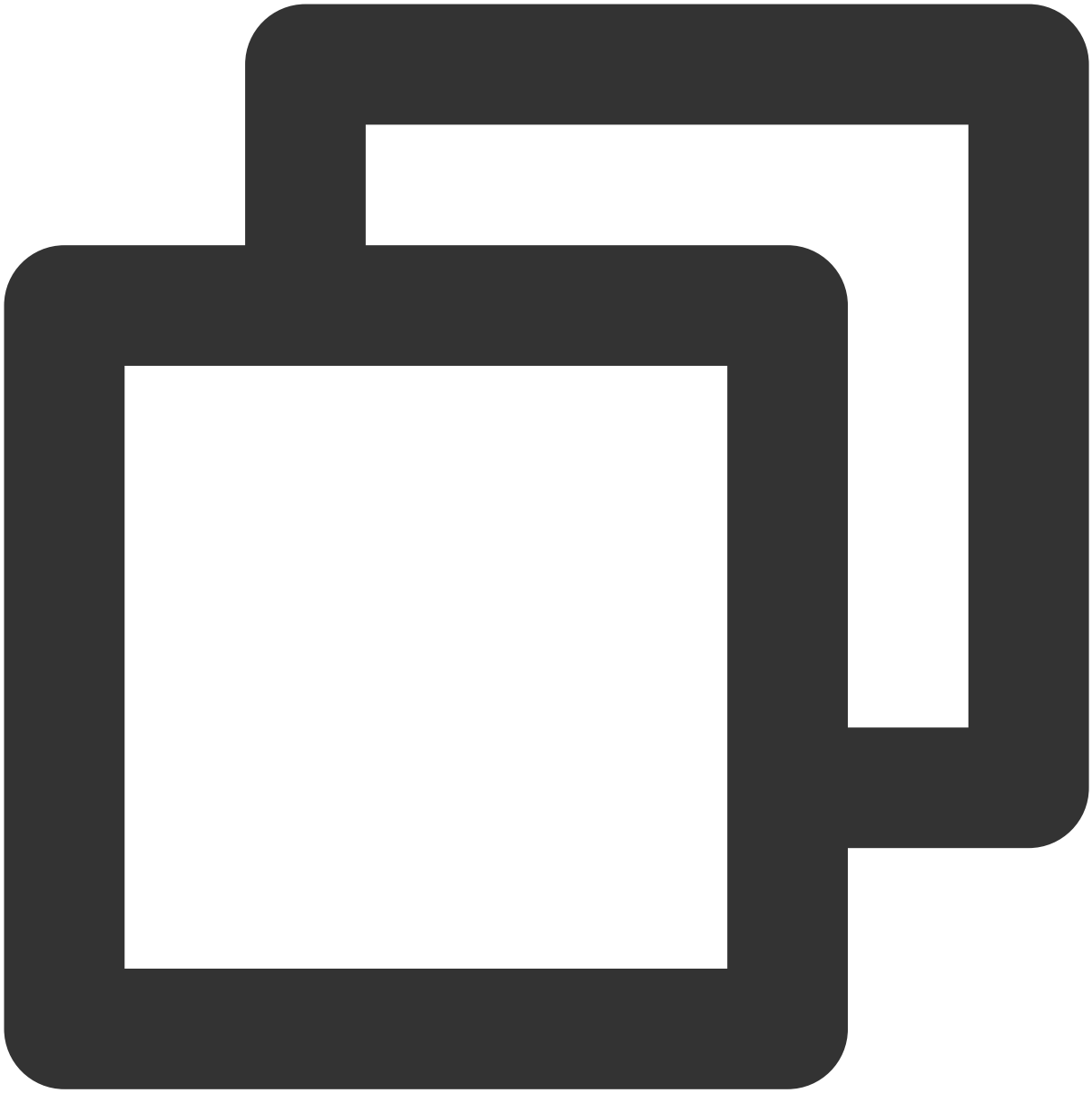
Scopes: http, server, location

3. To modify the configuration, save and exit the file, enter the Nginx bin file directory, and run the following command to reload Nginx:



```
./nginx -s reload
```

4. Use the following curl command to test whether Gzip compression is enabled:



```
curl -I -H "Accept-Encoding: gzip, deflate" "http://cloud.tencent.com/example/"
```

If a result is returned, Gzip compression is enabled.

If no result is returned, Gzip compression is not enabled.

HTTPS Forwarding Configurations

Last updated：2024-01-04 14:39:00

1. CLB Capability Description

By deeply optimizing the protocol stack and server, Tencent Cloud CLB achieves great improvement in HTTPS performance. Meanwhile, Tencent Cloud substantially reduces certificate costs through collaboration with international certificate authorities. CLB can bring significant benefits to 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 can sustain 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 costs, greatly improves service capability during business peaks and traffic surges, and strengthens the computation-based anti-attack capability.
3. Offloading and conversion of multiple protocols are supported, which reduces the business stress in adaption to various client protocols. The business backend only needs to support HTTP/1.1 to use different protocols such as HTTP/2, SPDY, SSL 3.0 and TLS 1.2.
4. One-stop SSL certificate application, monitoring, and replacement services are provided. Tencent Cloud cooperates with Comodo and SecureSite, two leading global certificate authorities, to simplify the certificate application process and reduce application costs.
5. Anti-CC and WAF features are provided to effectively defend against various attacks at the application layer, such as slow HTTP attacks, high-traffic DDoS attacks, SQL injections, and website trojans.

2. HTTP and HTTPS Headers

CLB acts as a proxy for HTTPS. When an HTTP or HTTPS request is forwarded to a real server by a CLB instance, you can set the protocol between the CLB instance and the real server to HTTP, HTTPS or gRPC. For more information, see Configuring an HTTPS Listener. When HTTP is selected, developers may not be able to distinguish whether the original request from the client is an HTTP or HTTPS request.
A CLB instance adds the `X-Client-Proto` header in a request before forwarding the request to a real server:
X-Client-Proto: http (The client request is an HTTP request.)
X-Client-Proto: https (The client request is an HTTPS request.)

3. Getting Started

Assume that you need to configure the website `https://example.com`, so that end users can visit it securely over HTTPS when they directly enter `www.example.com` in the browser.

For information about CLB operations, see the following documents:

Creating CLB Instances

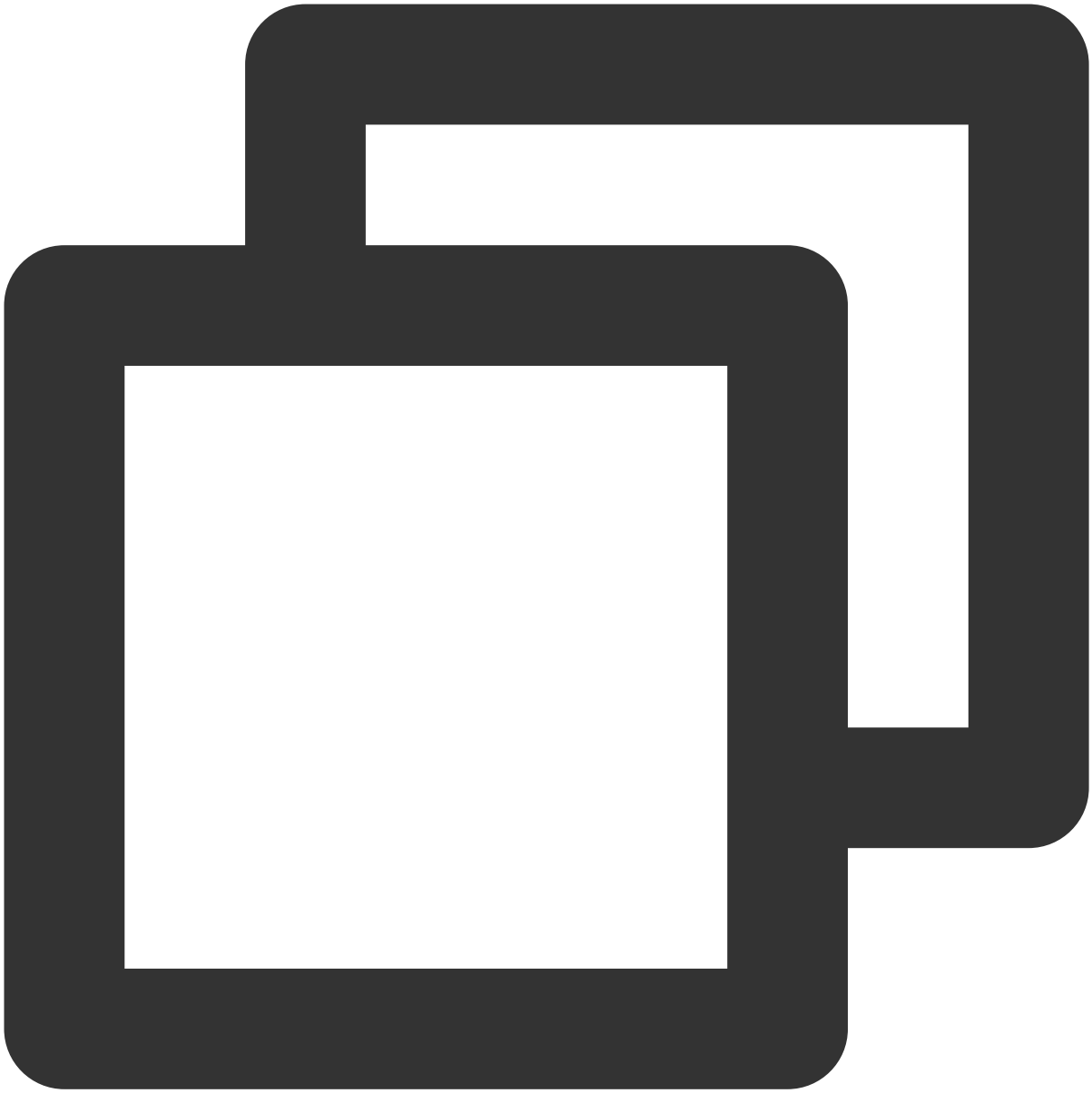
Configuring an HTTP Listener

Managing Real Servers

The request for accessing `www.example.com` entered by an end user is forwarded as below:

1. The request is transferred over HTTP and accesses port 80 of the CLB listener through VIP. Then, it is forwarded to port 8080 of the real server.
2. With the configuration of rewrite in Nginx on the real server, the request passes through port 8080 and is rewritten to the `https://example.com` page.
3. Then, the browser sends the `https://example.com` request to the corresponding HTTPS site again. The request accesses port 443 of the CLB listener through VIP and then is forwarded to port 80 of the real server.

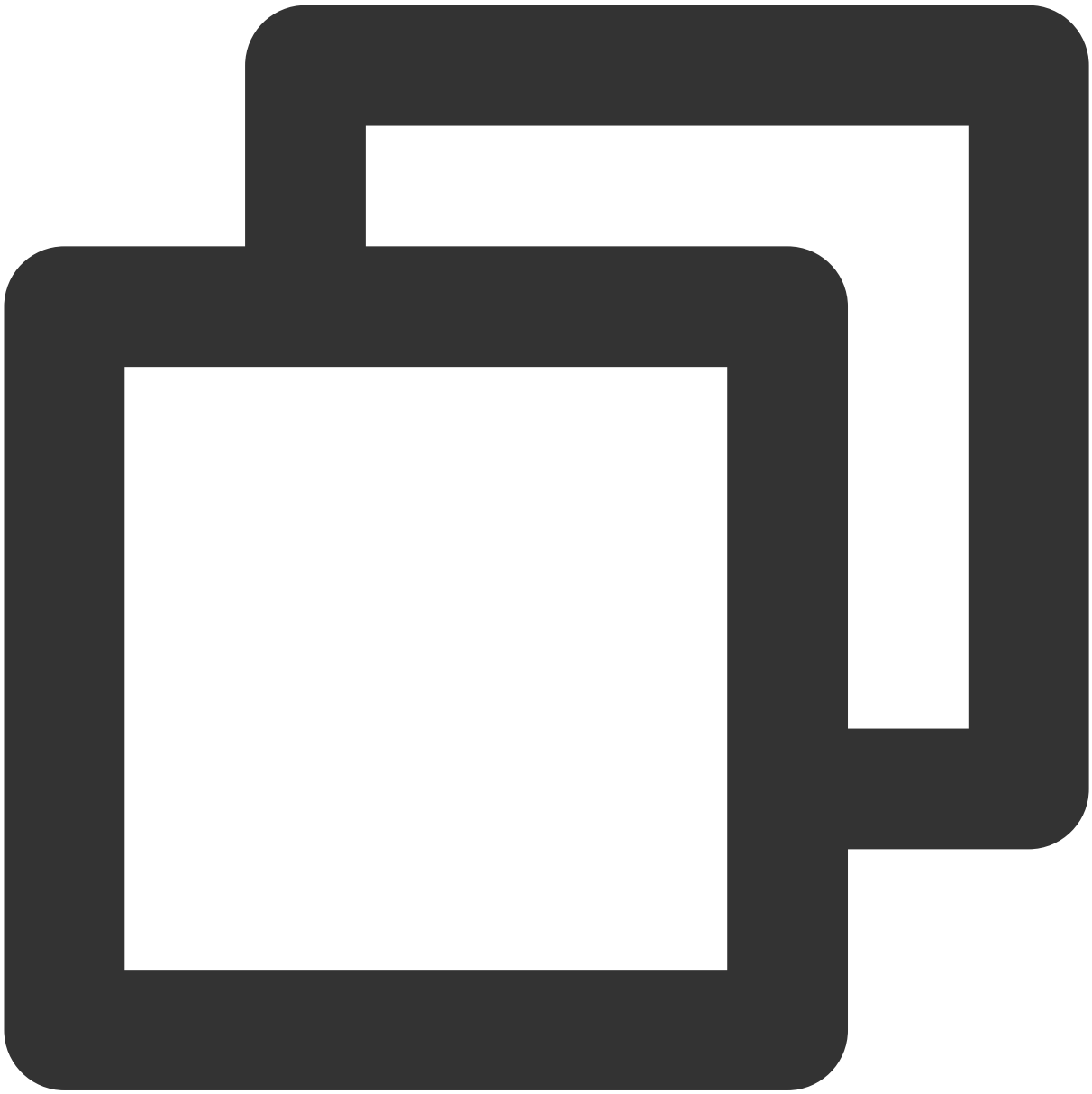
This operation rewrites a browser user's HTTP request to a more secure HTTPS request and is imperceptible to the user. To implement the above request forwarding operation, you can configure the real server as follows:



```
server {

    listen 8080;
    server_name example.qcloud.com;

    location / {

        #! customized_conf_begin;
        client_max_body_size 200m;
        rewrite ^/.(.*) https://$host/$1 redirect;
```

```
}
}
```

Alternatively, in the new version of Nginx, redirect the Nginx HTTP page to the HTTPS page by using the recommended 301 redirect:



```
server {
      listen  80;
      server_name    example.qcloud.com;
      return      301 https://$server_name$request_uri;
}
```

```
server {
      listen      443 ssl;
     server_name    example.qcloud.com;
    [....]
}
```

Obtaining Real Client IPs
Obtaining Real Client IPs in IPv4 CLB Scenarios

Last updated：2024-01-04 14:39:00

Notes on Getting Real Client IP Addresses by CLB

All layer-4 (TCP/UDP/TCP SSL) and layer-7 (HTTP/HTTPS) CLB services support getting a real client IP address directly on a backend CVM instance with no additional configuration required.
For layer-4 CLB, the source IP address obtained on the backend CVM instance is the client IP address.
For layer-7 CLB, when a non-persistent connection is used between the CLB instance and the real server, the source IP address obtained on the backend CVM instance is the client IP address; when a persistent connection is used between the CLB instance and the real server, the CLB instance does not pass through the source IP address. You can use the `X-Forwarded-For` or `remote_addr` field to directly get the client IP address. For the access logs of layer-7 CLB, see Configuring Access Logs.
Note:
For layer-4 CLB, the client IP address can be directly obtained with no additional configuration required on the backend CVM instance.
For other layer-7 load balancing services with SNAT enabled, you need to configure the backend CVM instance and then use `X-Forwarded-For` to get the real client IP address.
Below are commonly used application server configuration schemes.

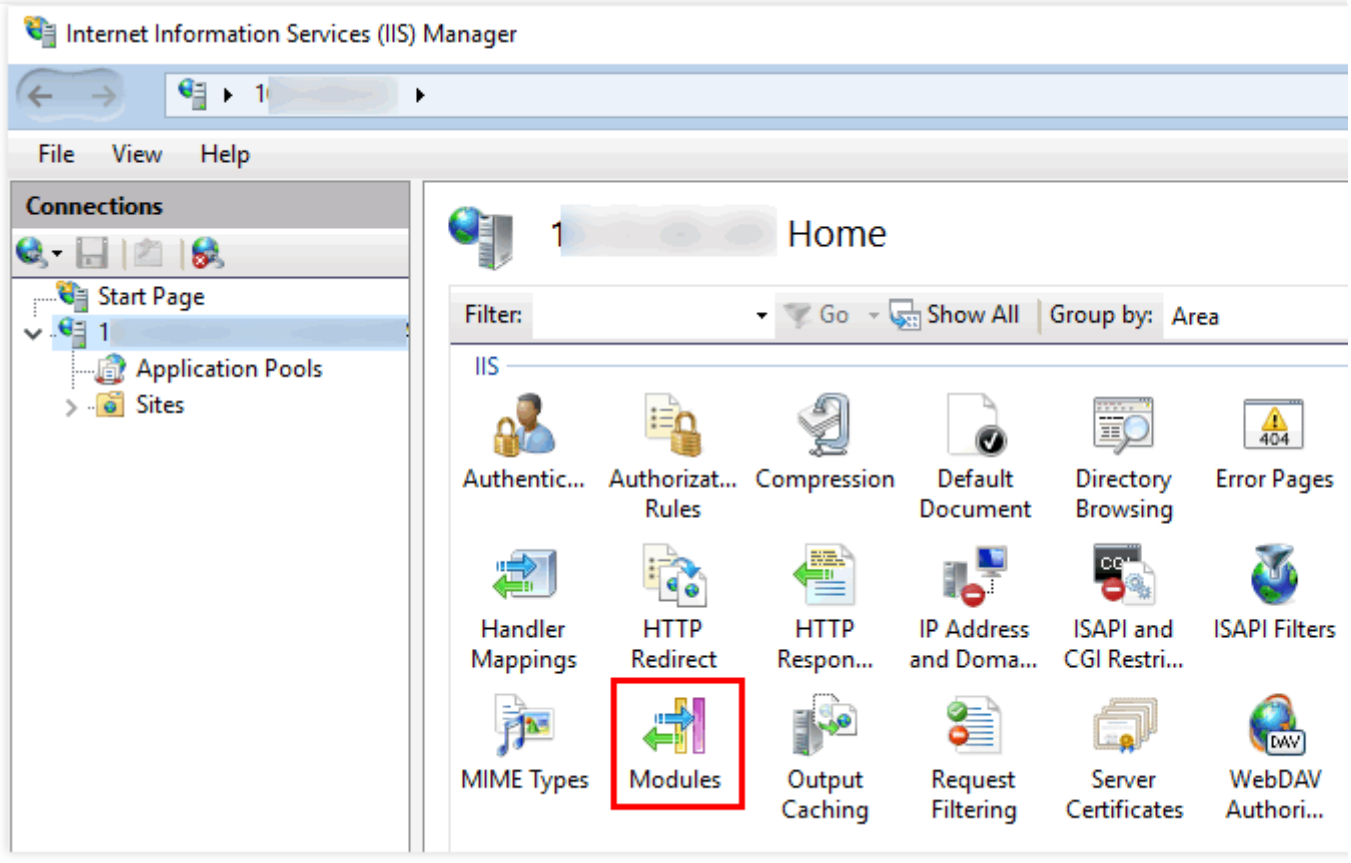
IIS 6 Configuration Scheme

1. Download and install the F5XForwardedFor plugin module, copy `F5XForwardedFor.dll` in the `x86\\Release` or `x64\\Release` directory based on your server operating system version to a certain directory (such as `C:\\ISAPIFilters` in this document), and make sure that the IIS process has the read permission on this directory.
2. Open **IIS Manager** and navigate to the web server you would like to apply it to. Right-click your web server and select **Properties**.
3. On the properties page, switch to **ISAPI Filters** and click **Add** to pop up the **Add/Edit Filter Properties** window.
4. In the pop-up window, enter "F5XForwardedFor" for **Filter Name** and the full path to `F5XForwardedFor.dll` for **Executable** and then click **OK**.

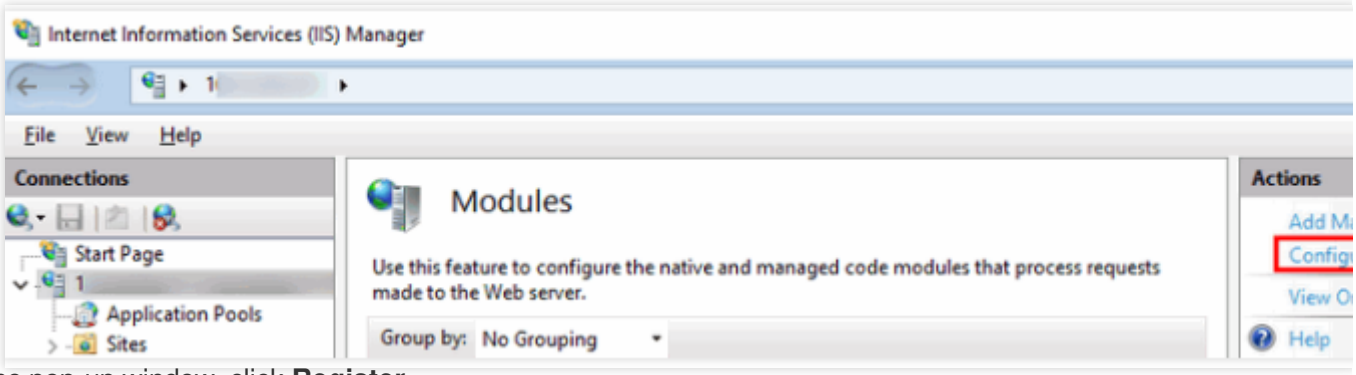
5. Restart the IIS server for the configuration to take effect.

IIS 7 Configuration Scheme

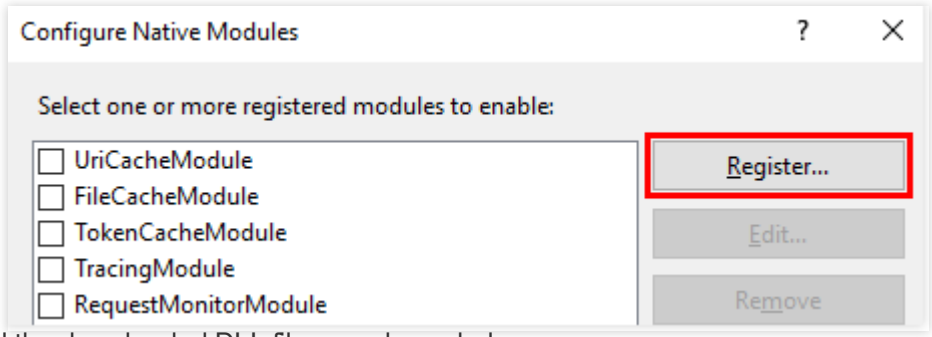
1. Download and install the F5XForwardedFor plugin module, copy `F5XFFHttpModule.dll` and `F5XFFHttpModule.ini` in the `x86\\Release` or `x64\\Release` directory based on your server operating system version to a certain directory (such as `C:\\x_forwarded_for` in this document), and make sure that the IIS process has the read permission on this directory.
2. Open **IIS Manager**, select your IIS server, and double-click **Modules**.



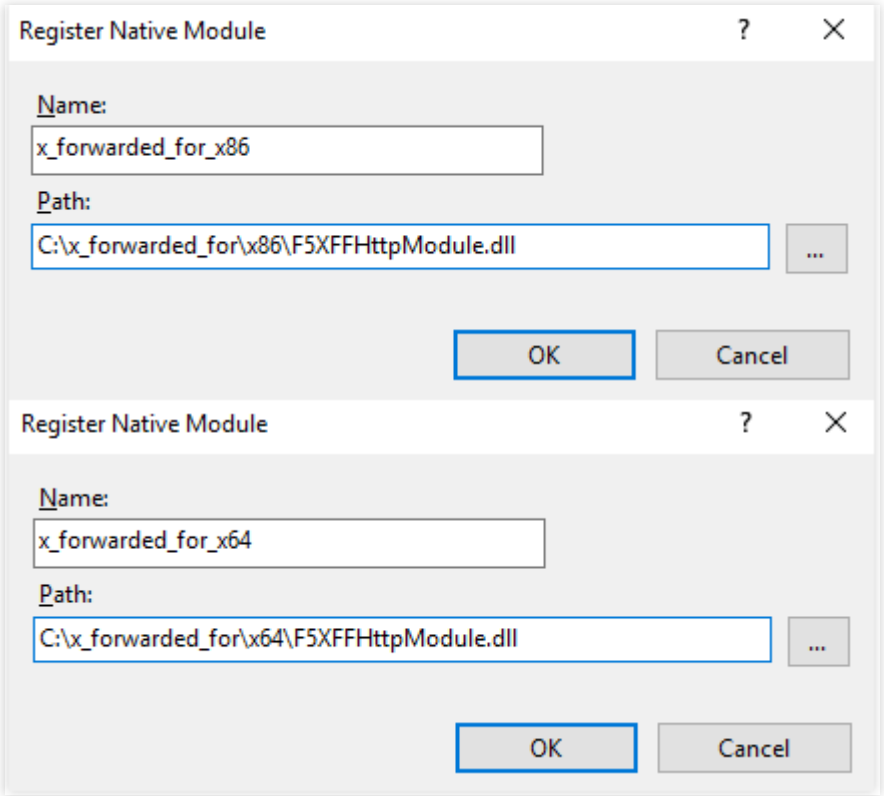
3. Click **Configure Native Modules**.



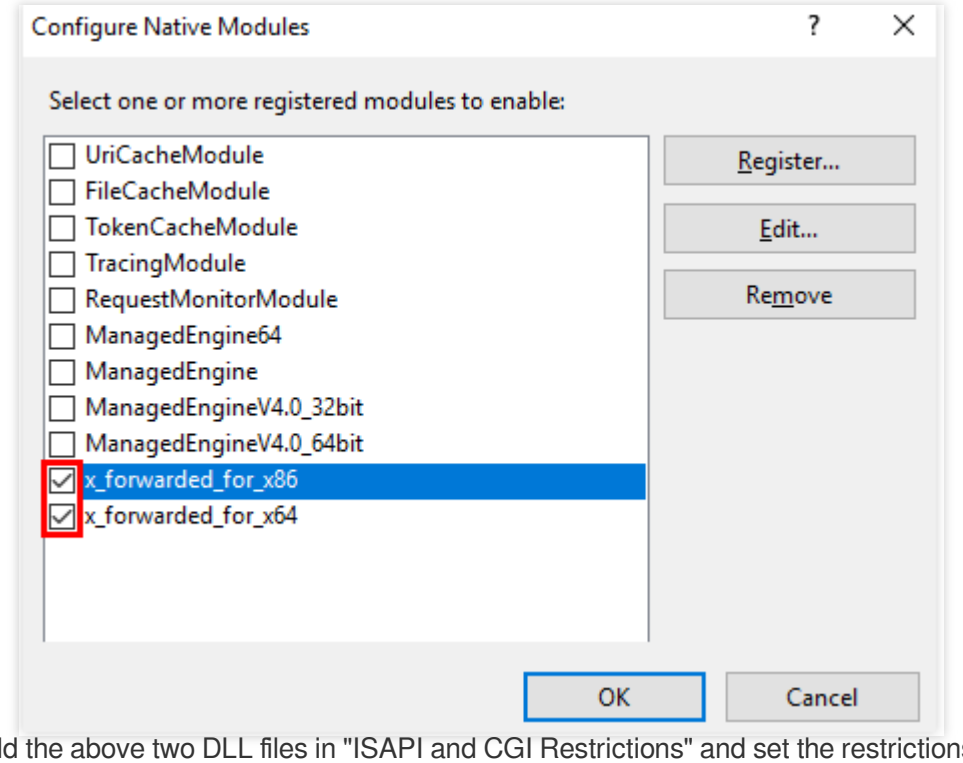
4. In the pop-up window, click **Register**.



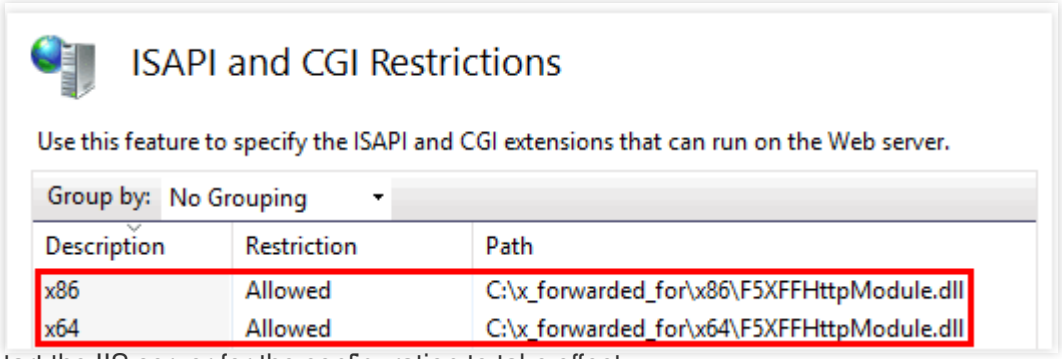
5. Add the downloaded DLL files, as shown below:



6. After adding the files, check them and click **OK**.



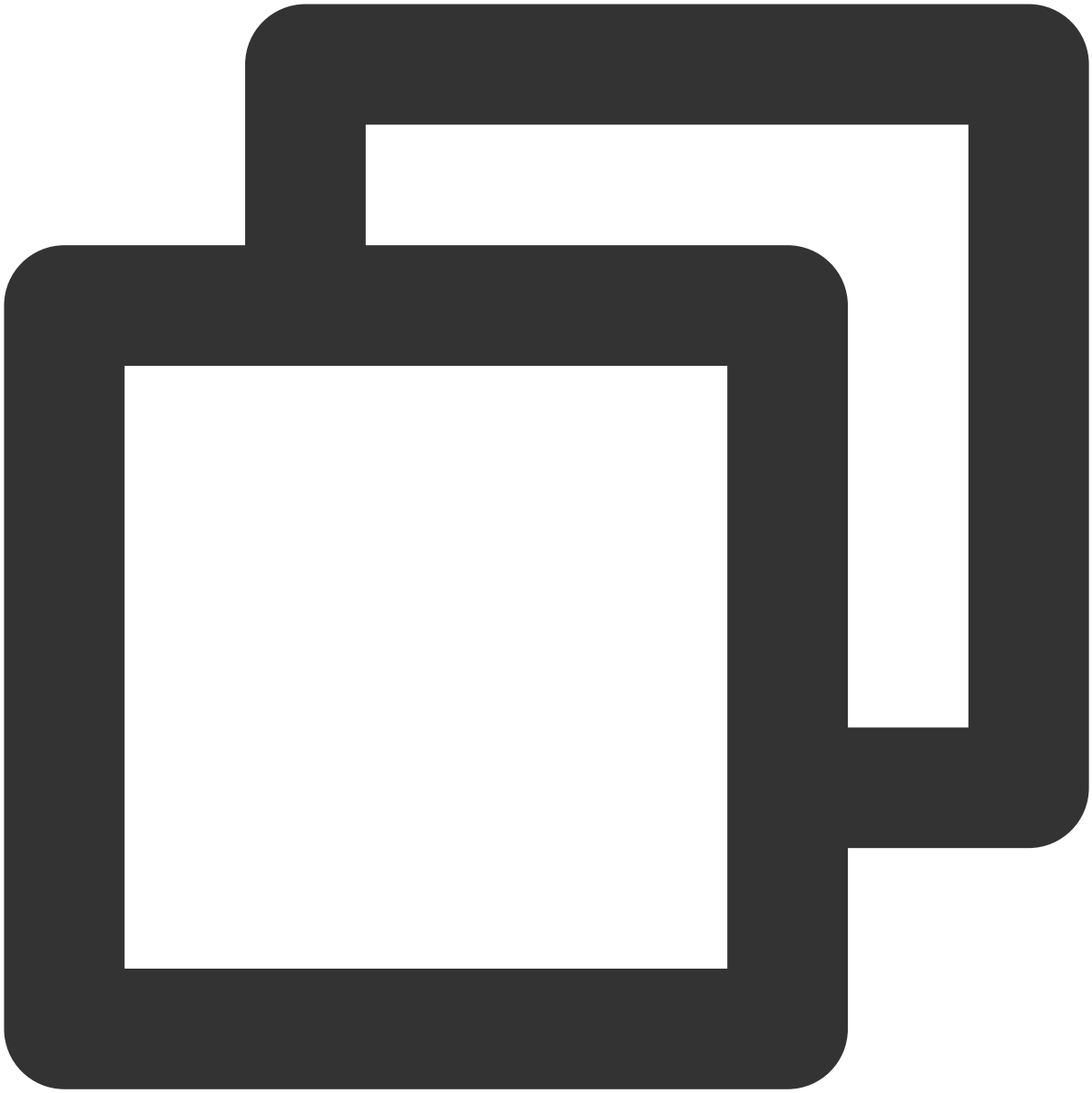
7. Add the above two DLL files in "ISAPI and CGI Restrictions" and set the restrictions to "Allow".



8. Restart the IIS server for the configuration to take effect.

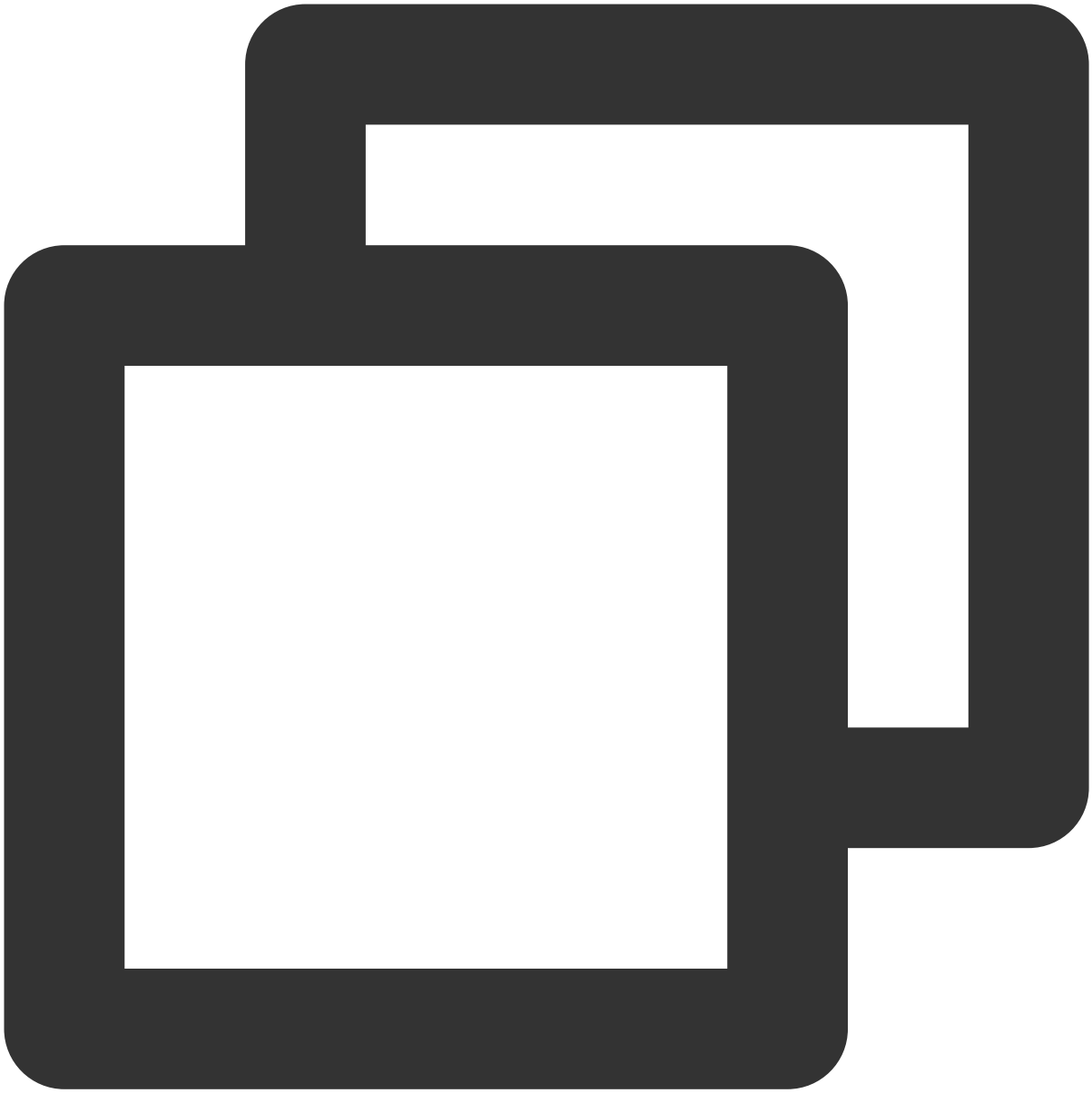
Apache Configuration Scheme

1. Install the third-party Apache module "mod_rpaf".



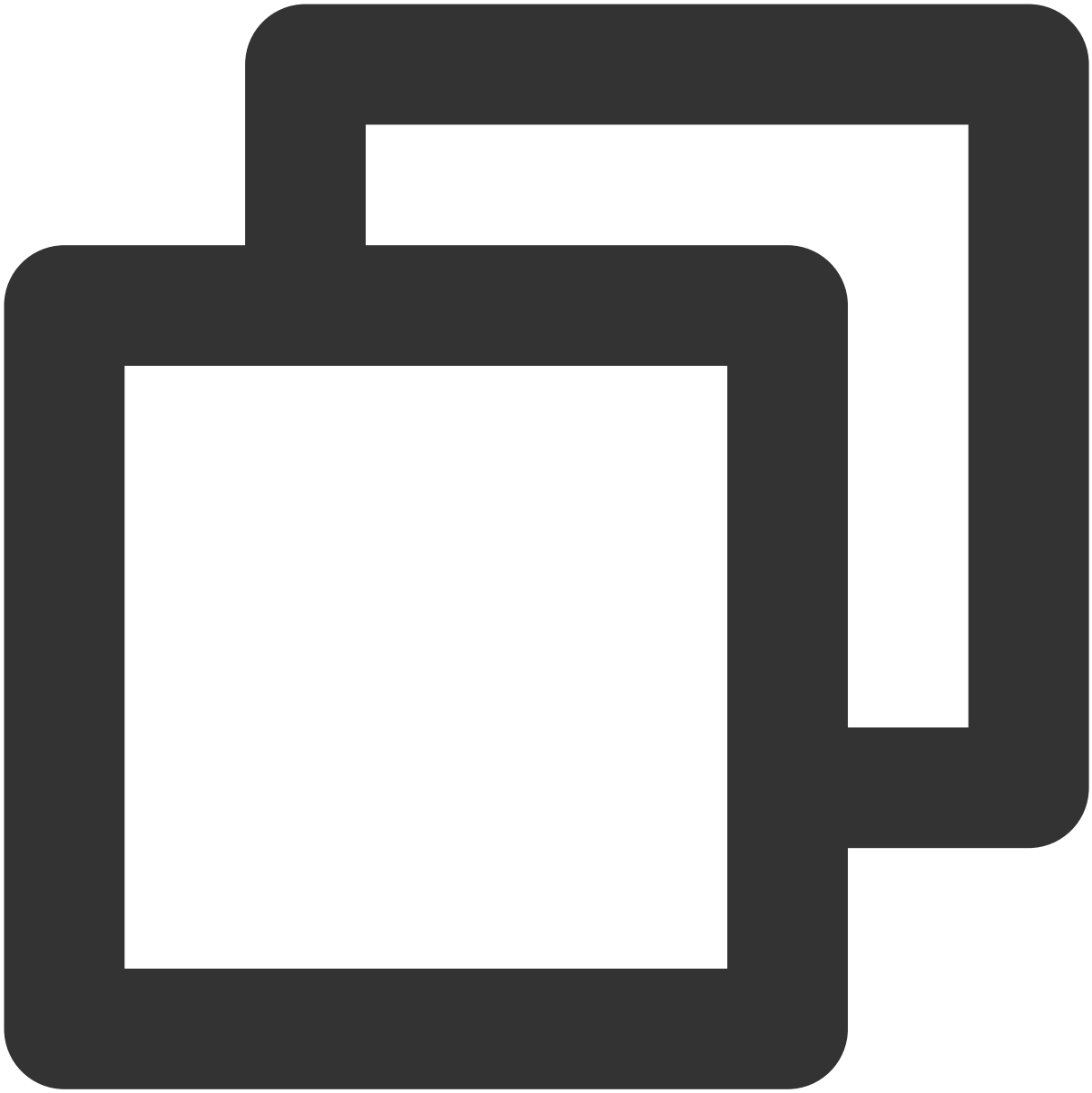
```
wget http://stderr.net/apache/rpaf/download/mod_rpaf-0.6.tar.gz
tar zxvf mod_rpaf-0.6.tar.gz
cd mod_rpaf-0.6
/usr/bin/apxs -i -c -n mod_rpaf-2.0.so mod_rpaf-2.0.c
```

2. Modify the Apache configuration file `/etc/httpd/conf/httpd.conf` by adding the following to the end of the file:



```
LoadModule rpaf_module modules/mod_rpaf-2.0.so
RPAFenable On
RPAFsethostname On
RPAFproxy_ips IP address (The IP address is not the public IP address provided by C
RPAFheader X-Forwarded-For
```

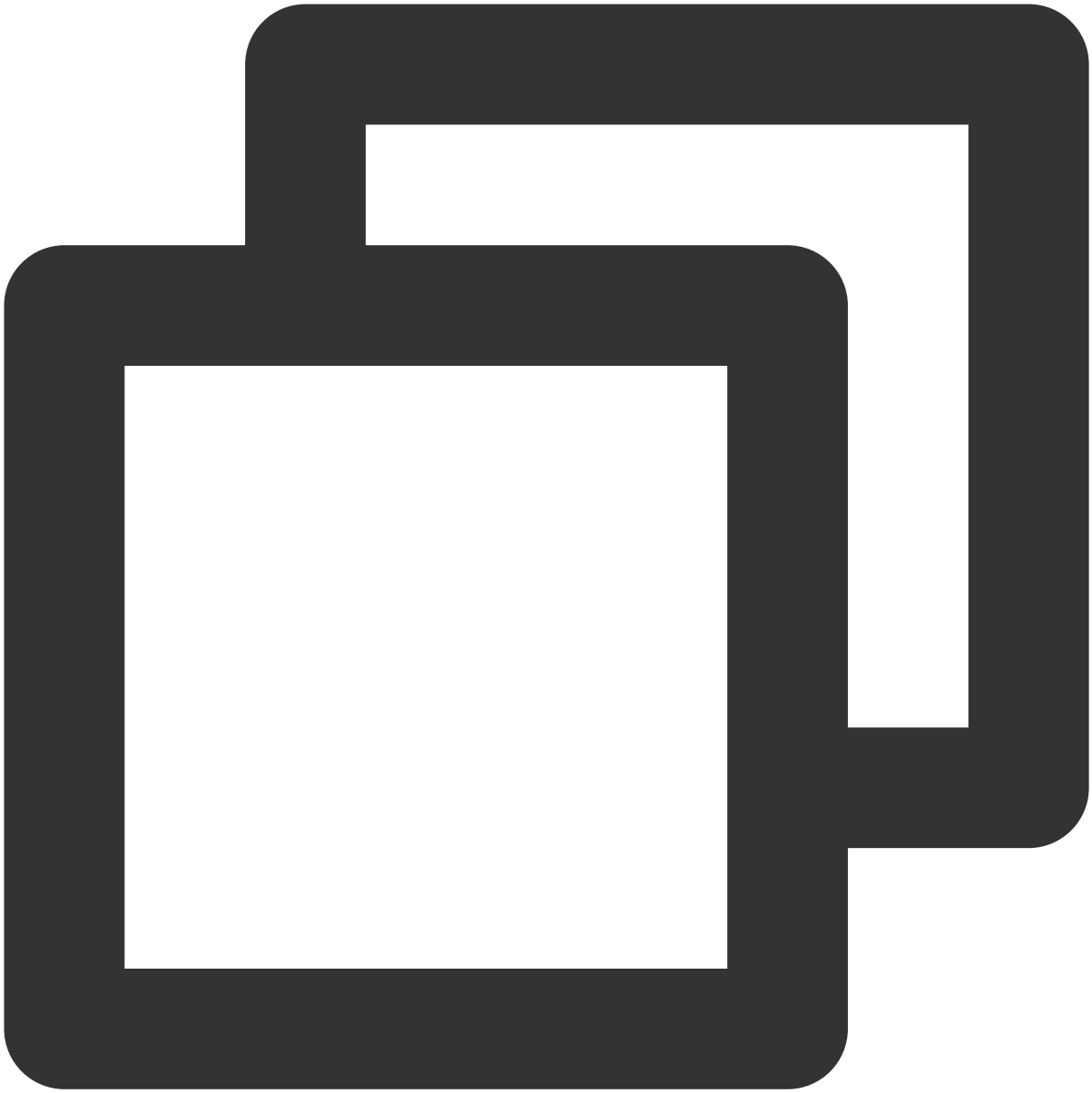
3. After adding the above content, restart Apache.



```
/usr/sbin/apachectl restart
```

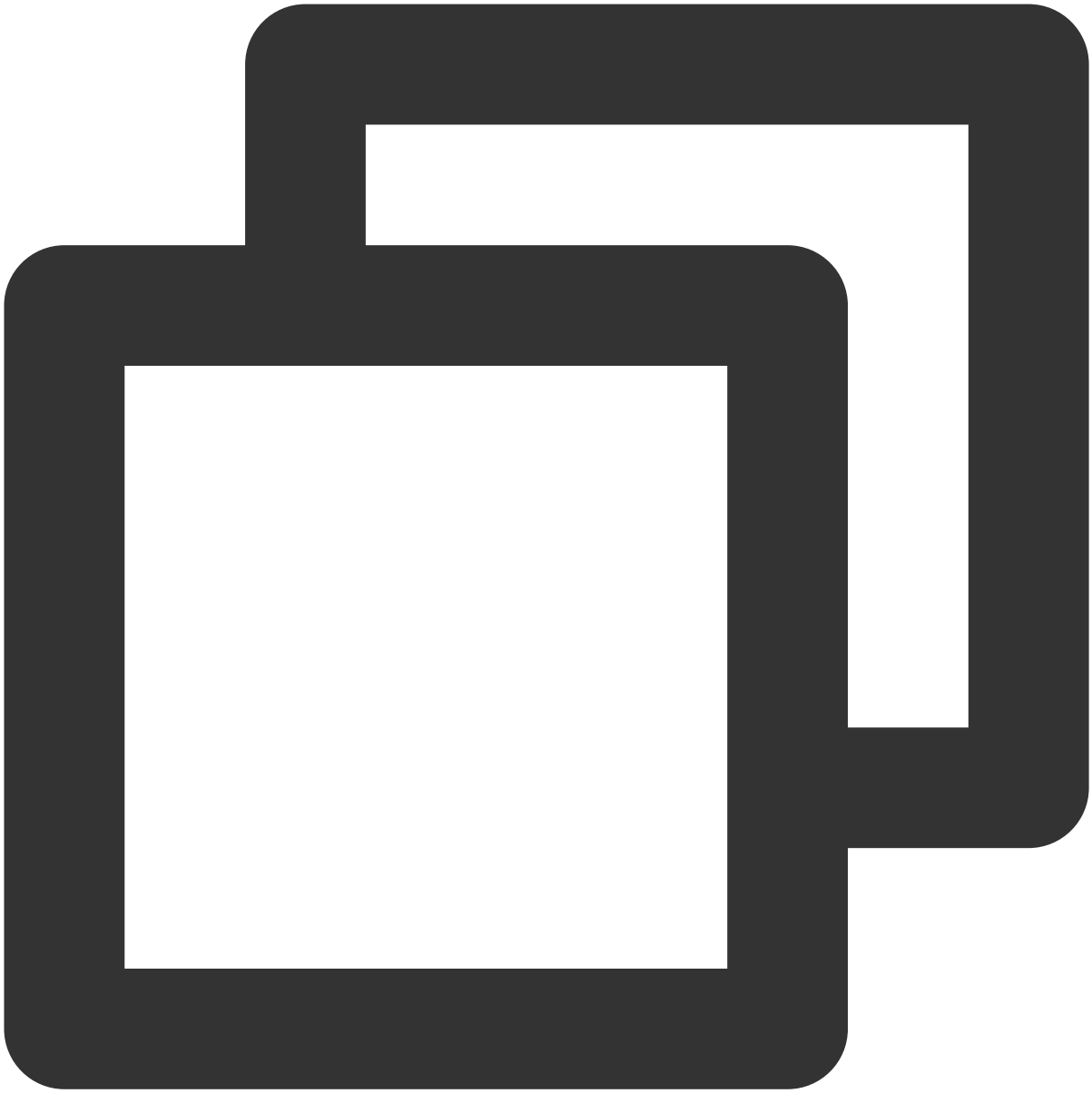
Nginx Configuration Scheme

1. You can use `http_realip_module` to get the real client IP address when Nginx is used as the server. However, this module is not installed in Nginx by default, and you need to recompile Nginx to add `--with-http_realip_module` .



```
yum -y install gcc pcre pcre-devel zlib zlib-devel openssl openssl-devel
wget http://nginx.org/download/nginx-1.17.0.tar.gz
tar zxvf nginx-1.17.0.tar.gz
cd nginx-1.17.0
./configure --prefix=/path/server/nginx --with-http_stub_status_module --without-ht
make
make install
```

2. Modify the `nginx.conf` file.

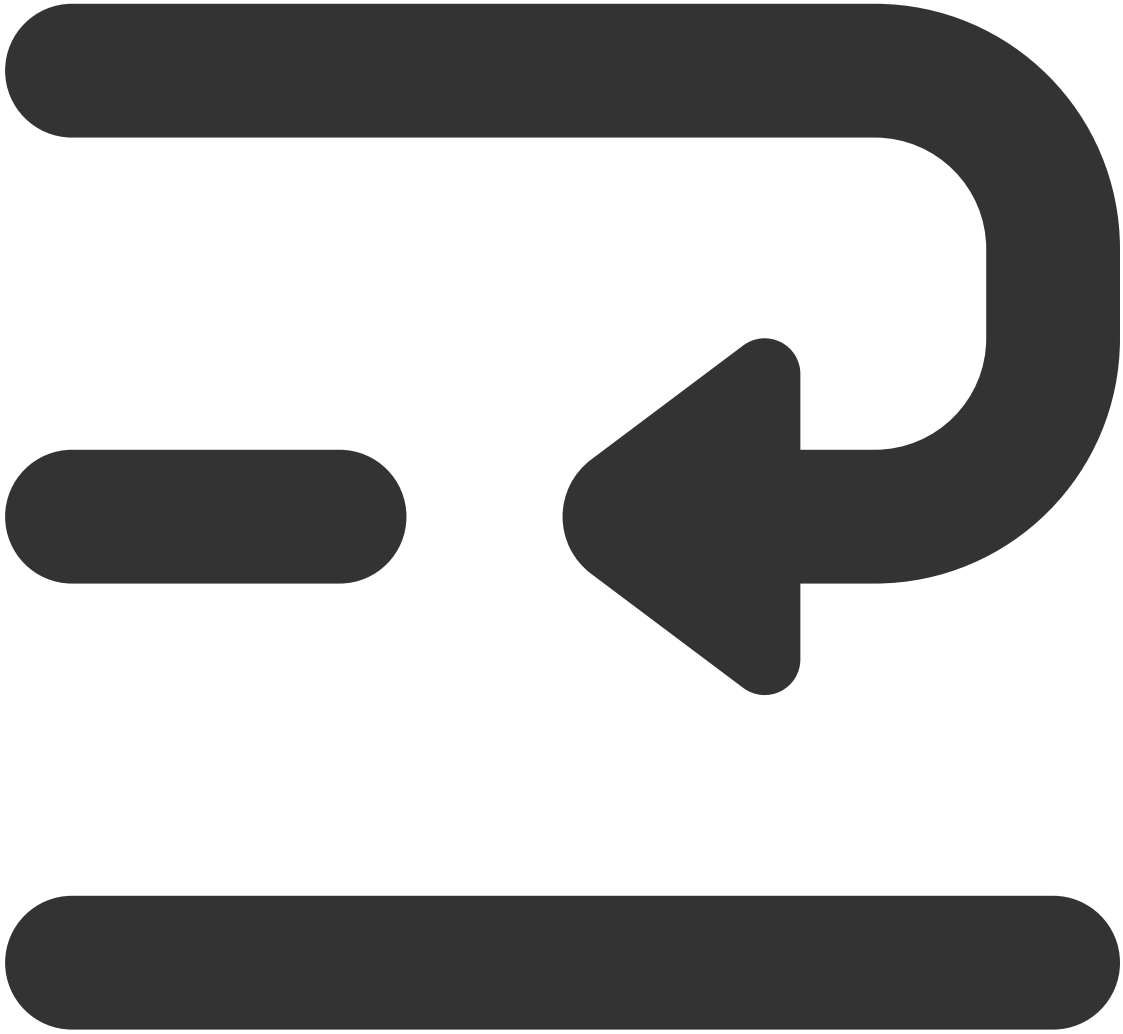


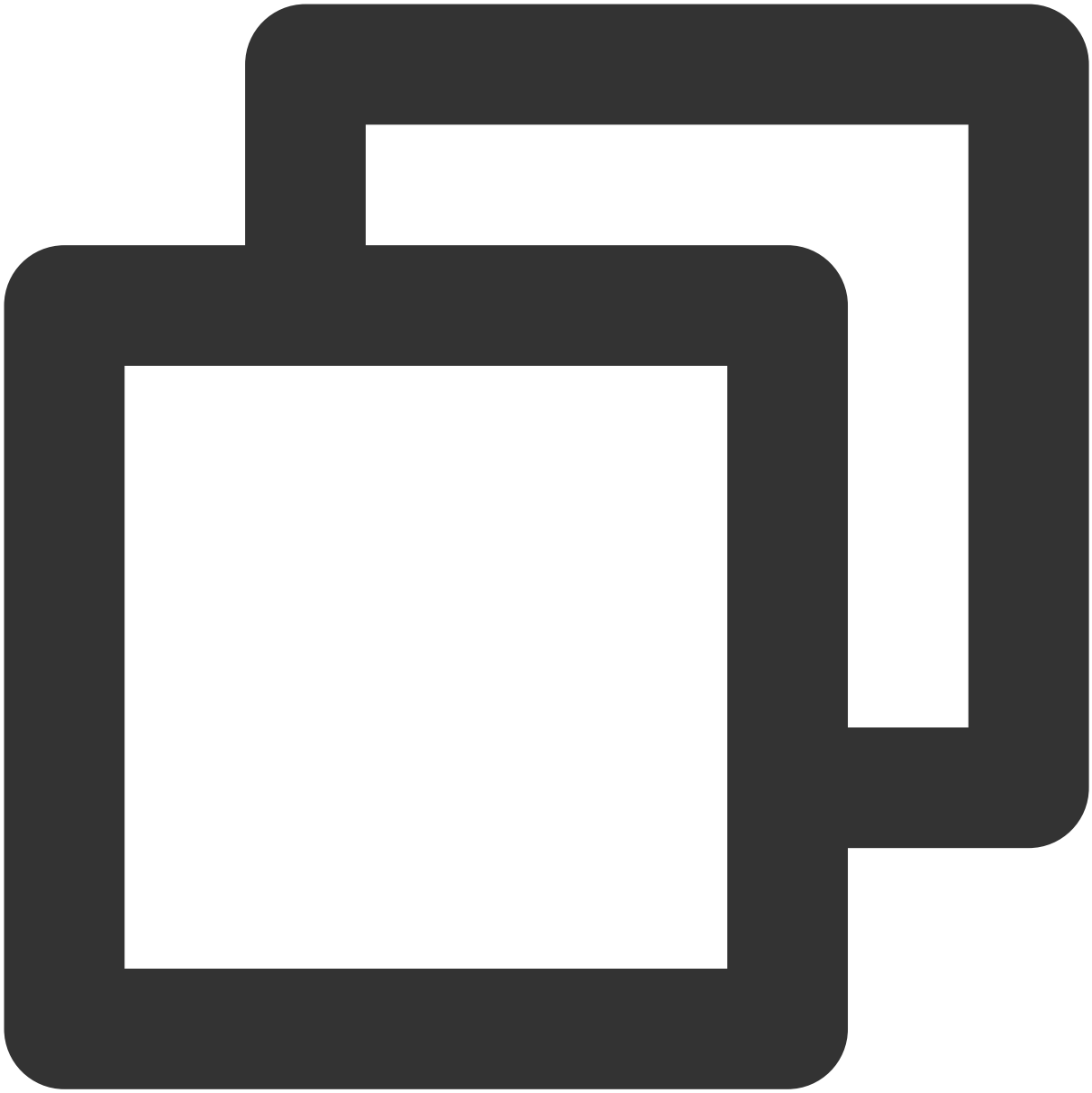
```
vi /etc/nginx/nginx.conf
```

Modify the configuration fields and information as follows:

Note:

You need to replace `xx.xx.xx.xx` with the IP or IP range of the upper-layer proxy server.

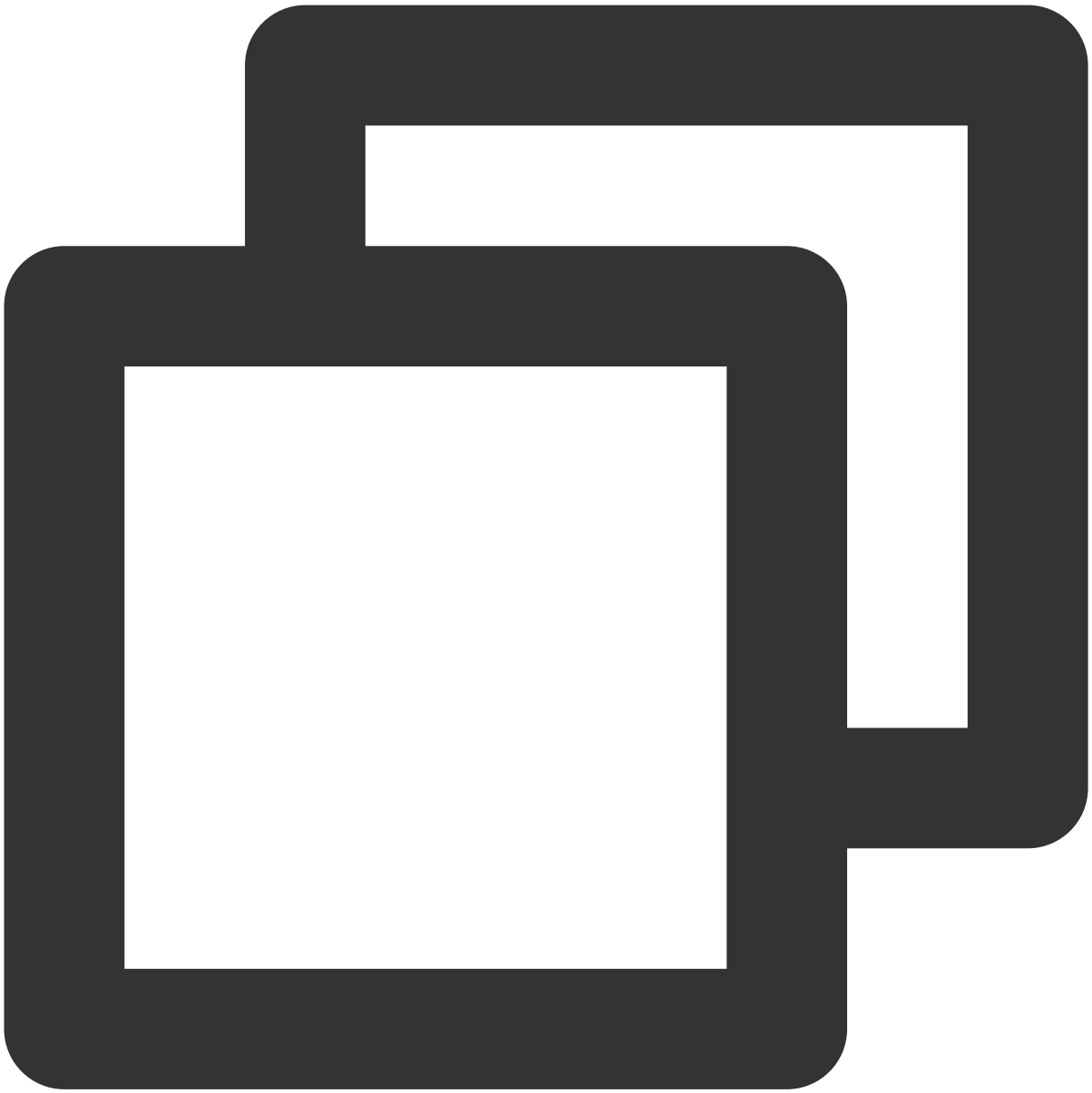




```
fastcgi connect_timeout 300;
fastcgi send_timeout 300;
fastcgi read_timeout 300;
fastcgi buffer_size 64k;
fastcgi buffers 4 64k;
fastcgi busy_buffers_size 128k;
fastcgi temp_file_write_size 128k;

# Modify the configuration fields and information as follows
set_real_ip_from xx.xx.xx.xx;
real_ip_header X-Forwarded-For;
```

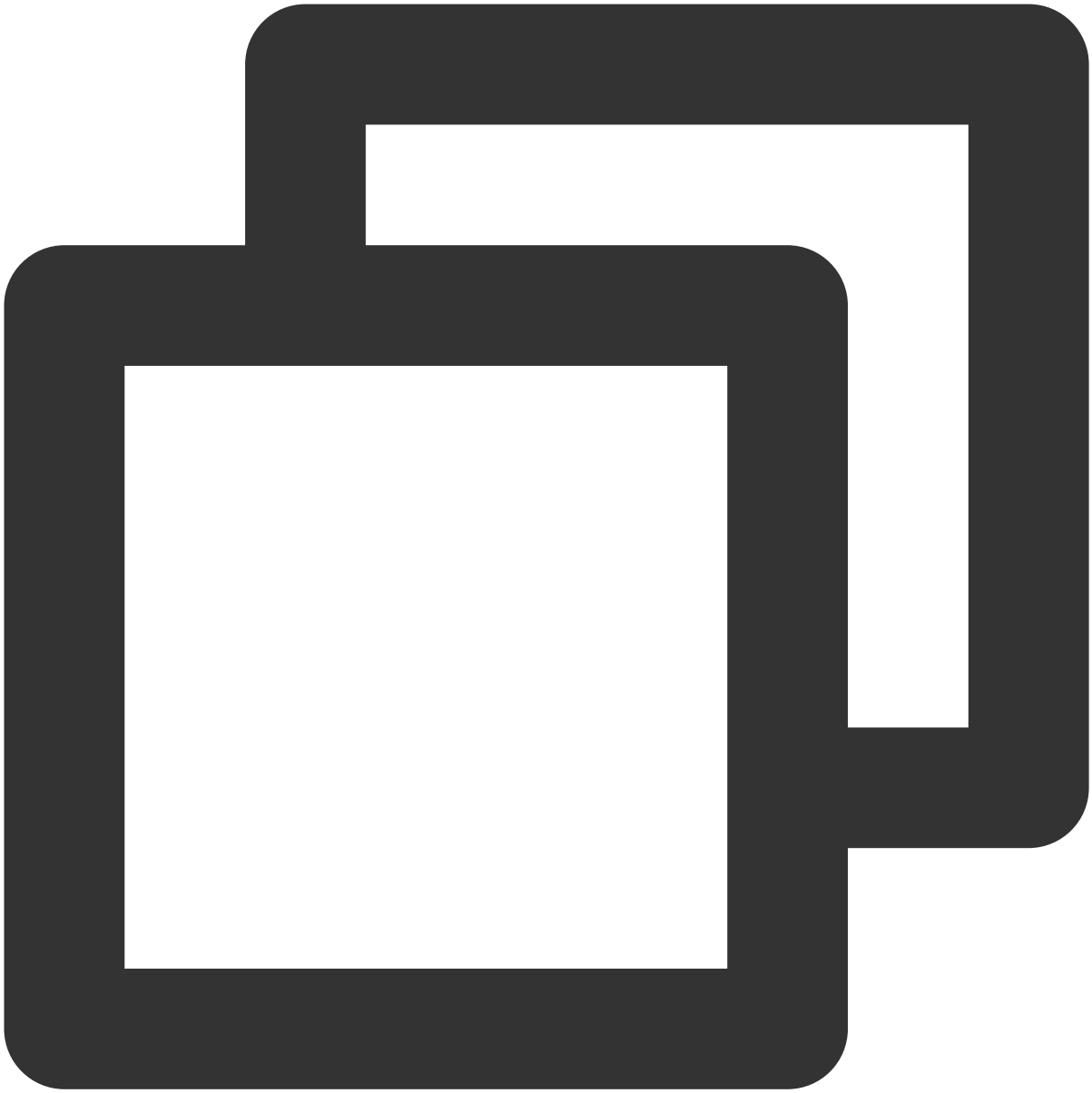
```
real_ip_recursive on;
```

3. Restart Nginx.



```
service nginx restart
```

4. View Nginx access logs to get the real client IP address.



```
cat /path/server/nginx/logs/access.log
```

Obtaining Real Client IPs via TOA in Hybrid Cloud Deployment

Last updated：2024-01-04 14:39:00

This document describes how the layer-4 (TCP) CLB service obtains the real client IP address via TOA in hybrid cloud deployment and NAT64 CLB scenarios.

Note:

Only NAT64 CLB instances in Beijing, Shanghai, and Guangzhou regions can obtain the real client IP address via TOA.

Only layer-4 (TCP) CLB instances can obtain the real client IP address via TOA, while layer-4 (UDP) and layer-7 (HTTP/HTTPS) CLB instances cannot.

This feature is in beta testing. To try it out, please submit a ticket.

Use Cases

Hybrid cloud deployment

In hybrid cloud deployment scenarios, IP addresses of the IDC and VPC may overlap, so an SNAT IP address is required. For the server, the real client IP address is invisible and needs to be obtained via TOA.

NAT64 CLB

In NAT64 CLB scenarios, the real client IPv6 address is translated to a public IPv4 address, which is invisible to the real server.

In this case, the real client IP address can be obtained via TOA, that is, the TCP packets transmit the real client IP address to the server after you insert the real client IP address into the field `TCP option`, and the client can obtain the real client IP address by calling the API of the TOA kernel module.

Restrictions

Resource limits

The kernel version of the TOA compilation environment must be consistent with that of the service environment.

In a container environment, the TOA kernel module needs to be loaded in the host.
To load the TOA kernel module, the root permission is required.
Compatibility limits
Compatibility limitsUDP listeners cannot obtain the real client IP address via TOA.
If TOA-related operations are already performed on devices between the client and the real server, the real server may fail to obtain the real client IP address.
After TOA is inserted, it takes effect only on new connections.
TOA may degrade the server performance as it needs to perform additional processing such as extracting the address from the field `TCP option` .
Compatibility issues may come up when Tencent Cloud TOA is used with other user-defined TOA modules.
Tencent Cloud TOA is embedded in TencentOS and can be used to obtain the real client IP address in hybrid cloud deployment scenarios. If the server is run on TencentOS and deployed in a hybrid cloud, you can directly run the command `modprobe toa` to load TOA. If the server is run on Linux, Linux TOA should be used instead.

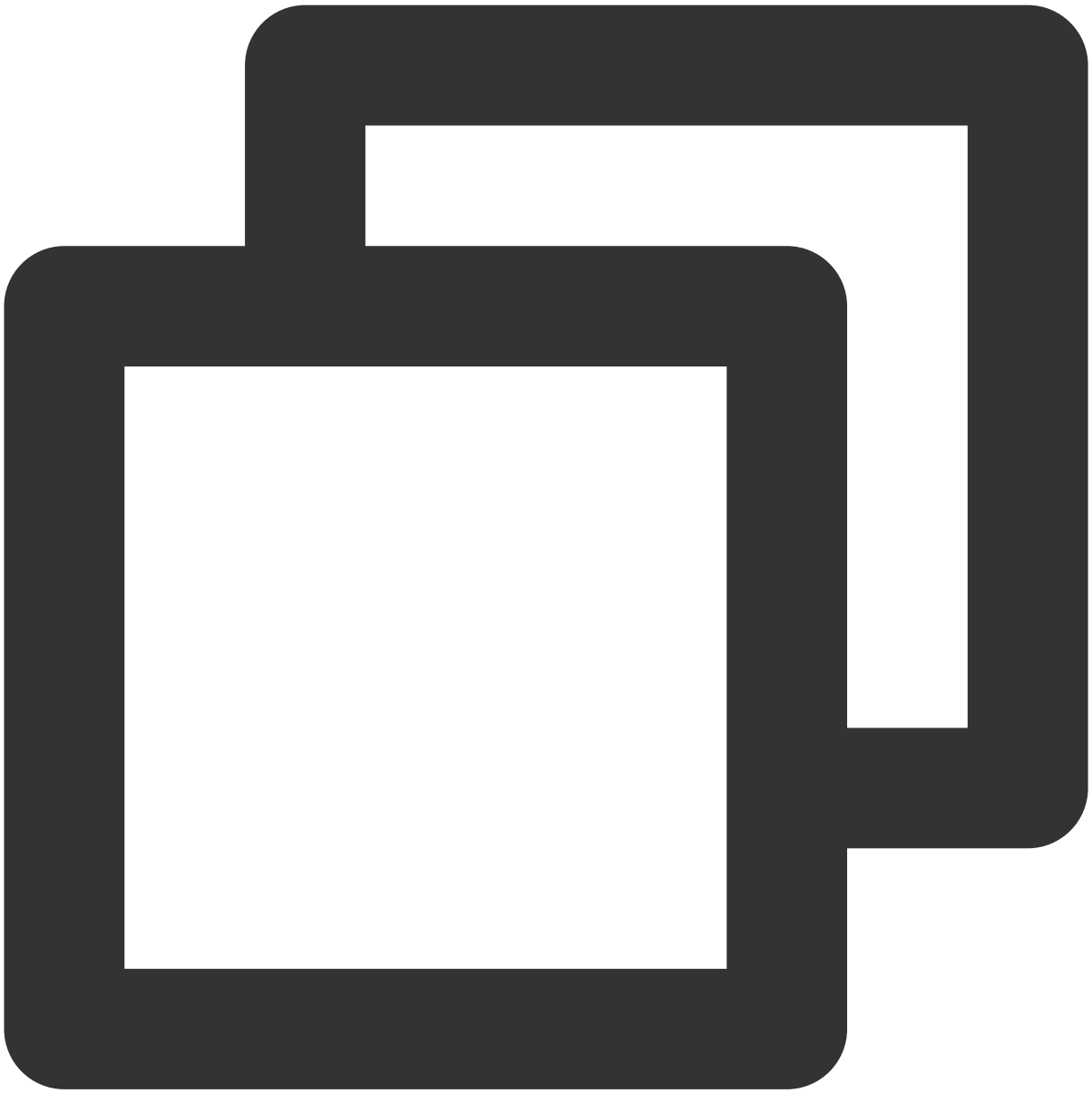
Enabling TOA in the Console

1. Create a NAT64 CLB instance. For more information, see Creating IPv6 NAT64 CLB Instances.
2. Log in to the CLB console and create a TCP listener. For more information, see Configuring TCP Listener.
3. Enable TOA in the **Create listener** window.

Loading TOA

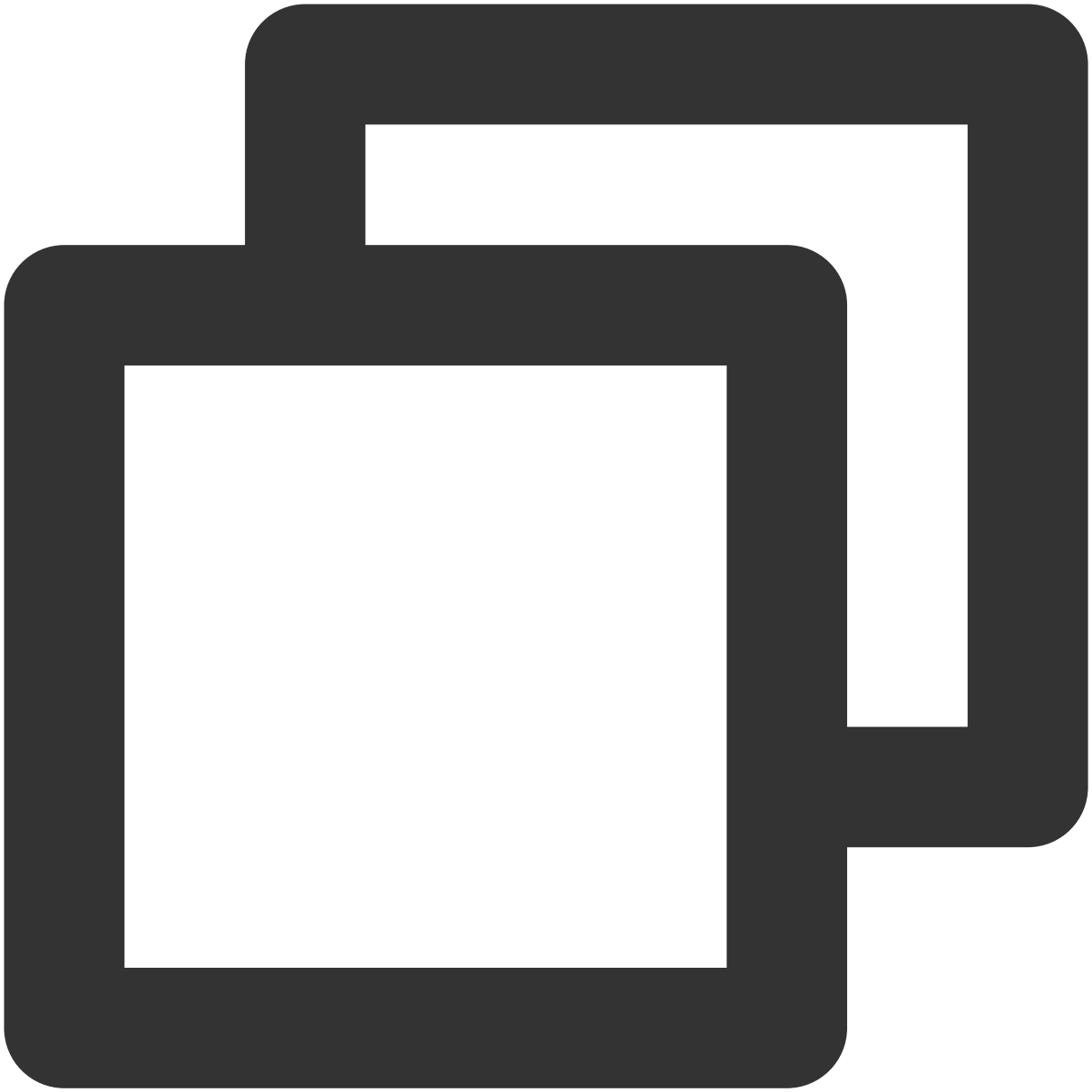
1. Download and decompress the TOA package corresponding to the version of Linux OS on Tencent Cloud.
centos
CentOS 8.0 64
CentOS 7.6 64
CentOS 7.2 64
debian
Debian 9.0 64
suse linux
SUSE 12 64
SUSE 11 64
ubuntu
Ubuntu 18.04.4 LTS 64
Ubuntu 16.04.7 LTS 64

2. After decompression is completed, run the `cd` command to access the decompressed folder and run the following module loading command:



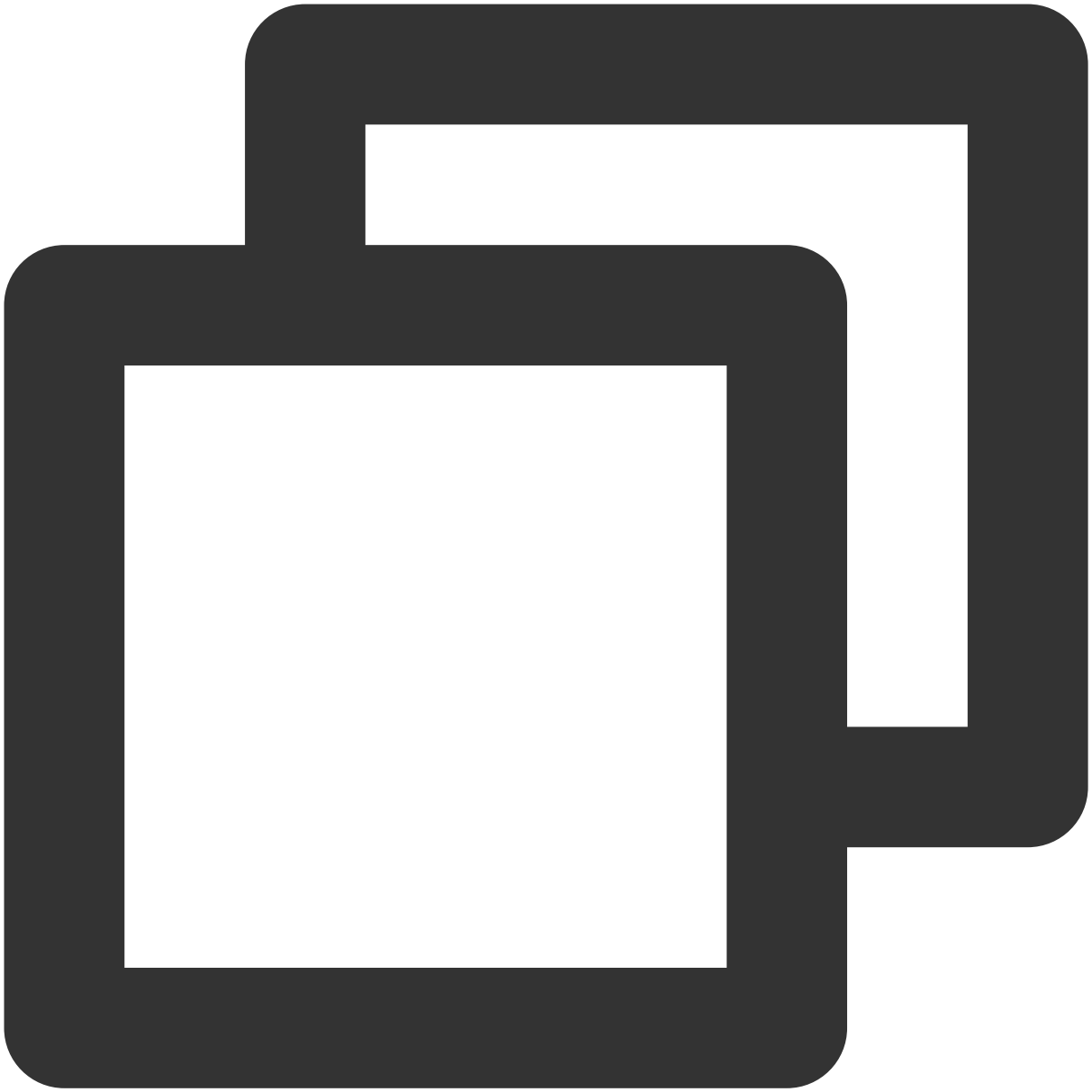
```
insmod toa.ko
```

3. Run the following command to check whether TOA has been loaded. If you see the message "toa load success", the loading is successful.



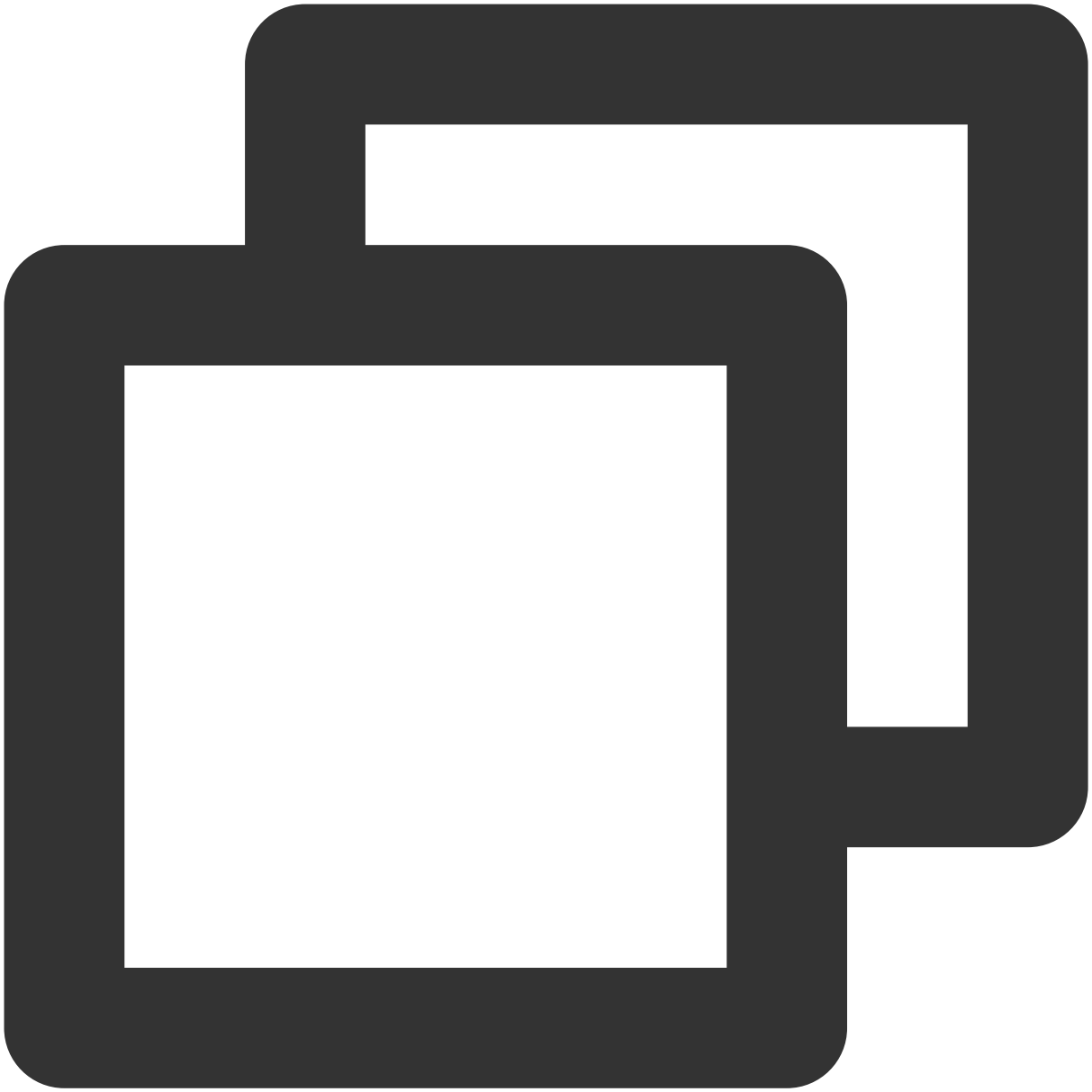
```
dmesg -T | grep TOA
```

4. After TOA is loaded, load the `toa.ko` file in the startup script (the `toa.ko` file needs to be reloaded if the server is restarted).
5. (Optional) If TOA is no longer needed, run the following command to uninstall it:



```
rmmod toa
```

6. (Optional) Run the following command to check whether the module is uninstalled. If you see the message "TOA unloaded", the uninstallation is successful.



```
dmesg -T
```

If you cannot find an installation package above for your OS version, you can download the general source package for Linux OS and compile it to obtain the `toa.ko` file. This general version supports most Linux distributions (e.g., CentOS 7, CentOS 8, Ubuntu 16.04, and Ubuntu 18.04).

Note:

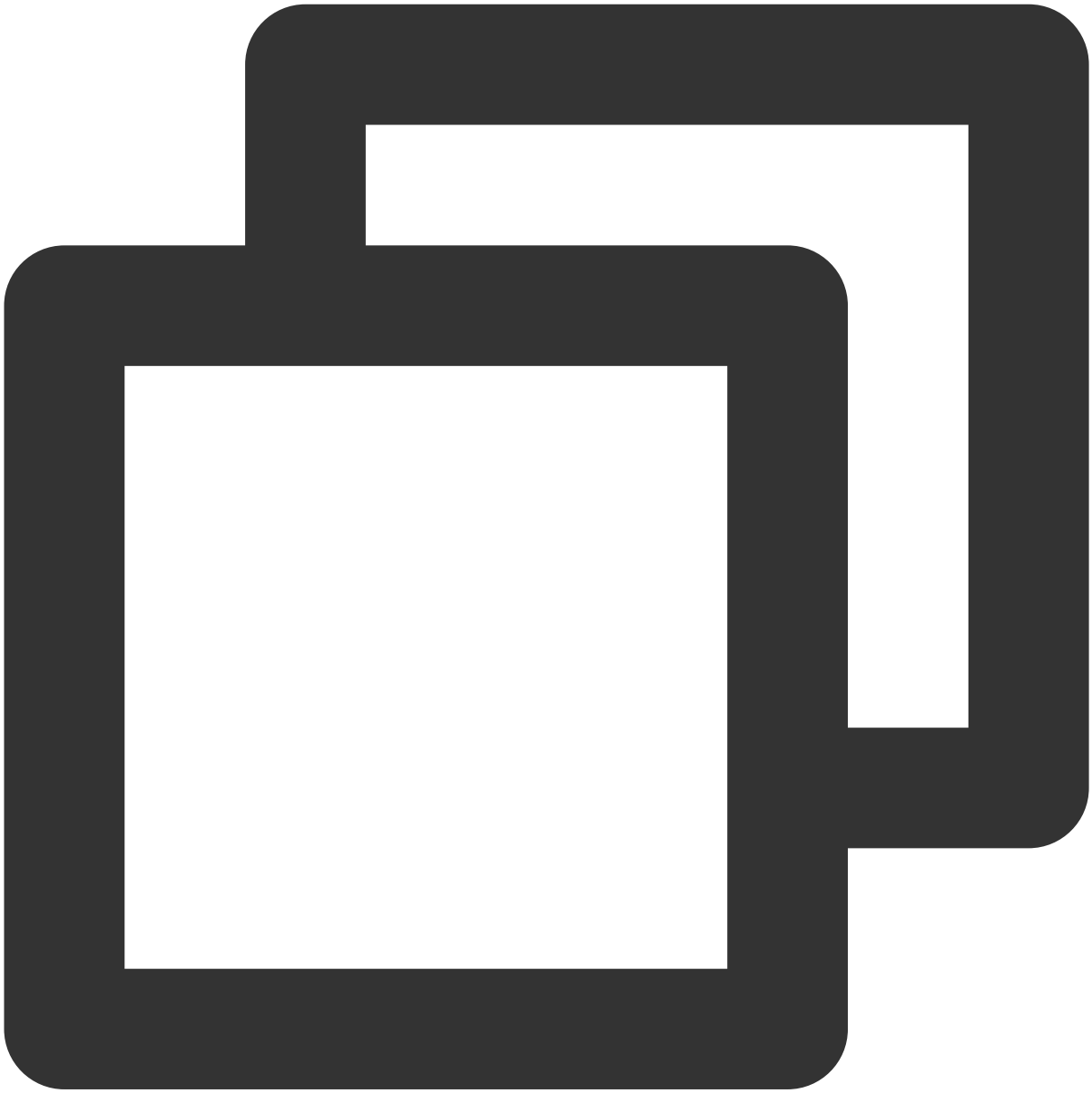
Linux kernels and Linux distributions are varied, and that may cause compatibility issues. We recommend compiling the TOA source package on your OS before using it.

1. Download the source package.

Note:

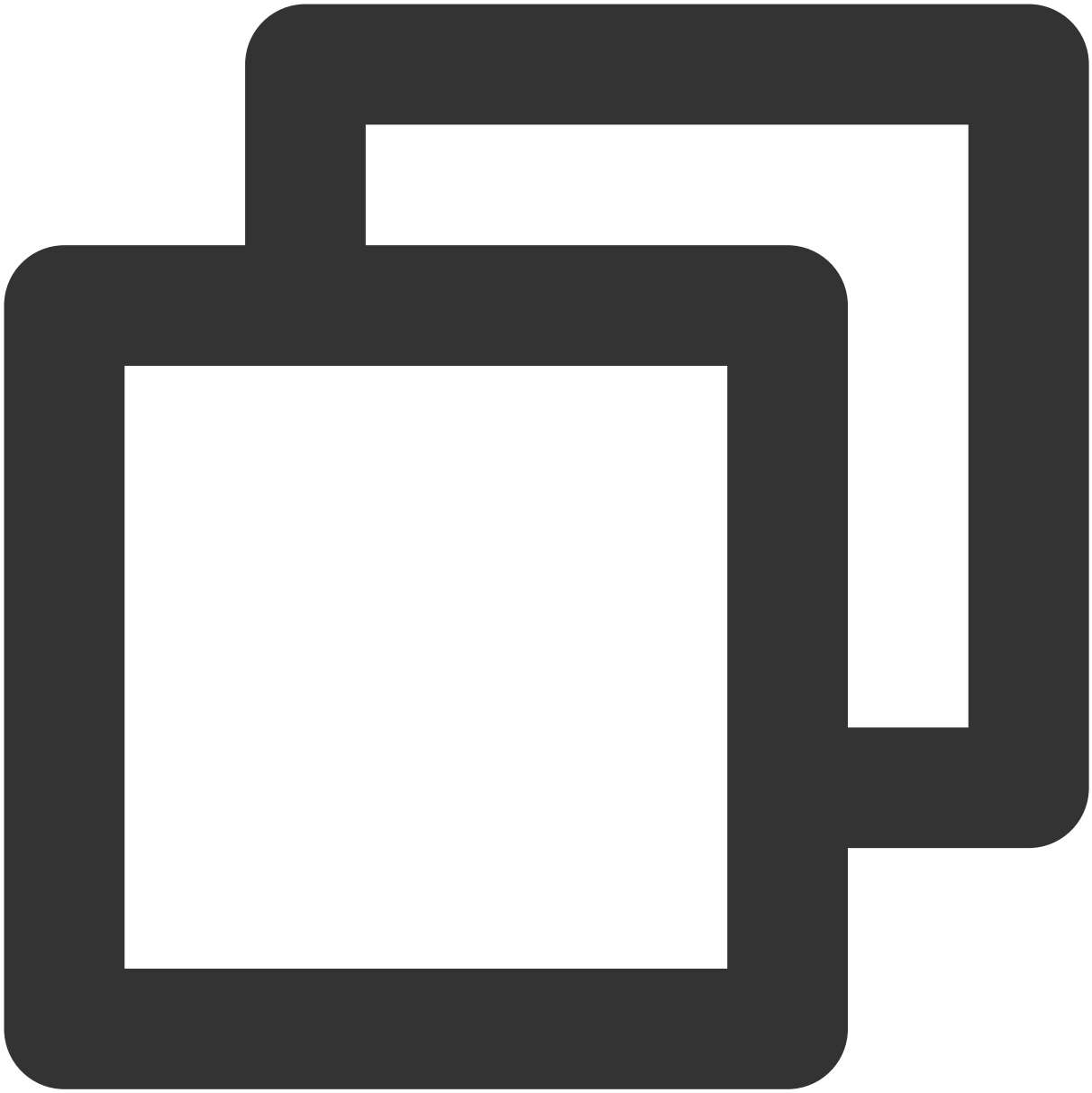
If your OS is Linux, download the Linux TOA source package; if it is TencentOS, download the TLinux TOA source package.

Linux



```
wget "https://clb-toa-1255852779.file.myqcloud.com/tgw_toa_linux_ver.tar.gz"
```

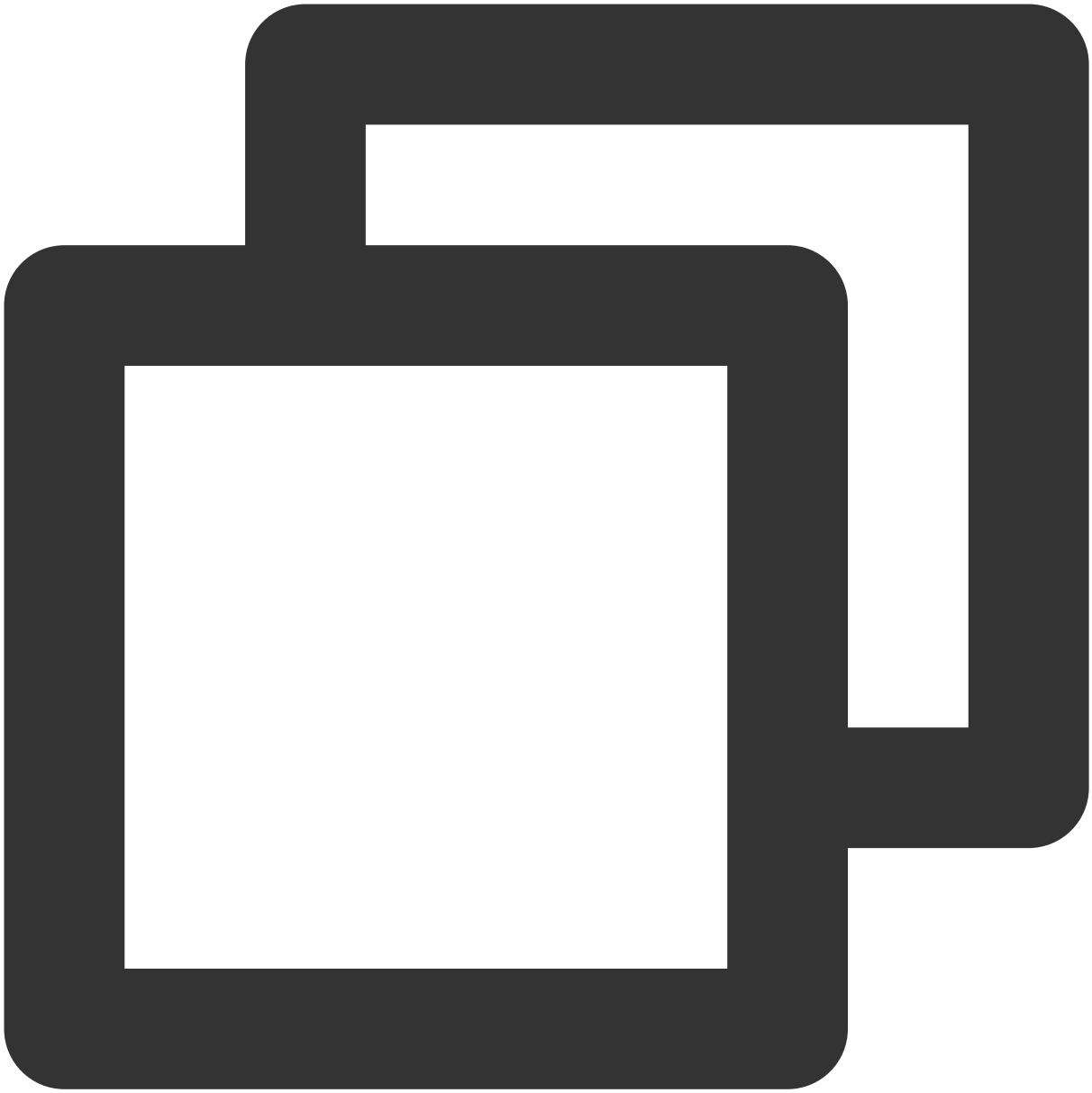
TLinux



```
wget "https://clb-toa-1255852779.file.myqcloud.com/tgw_toa_tlinux_ver.tar.gz"
```

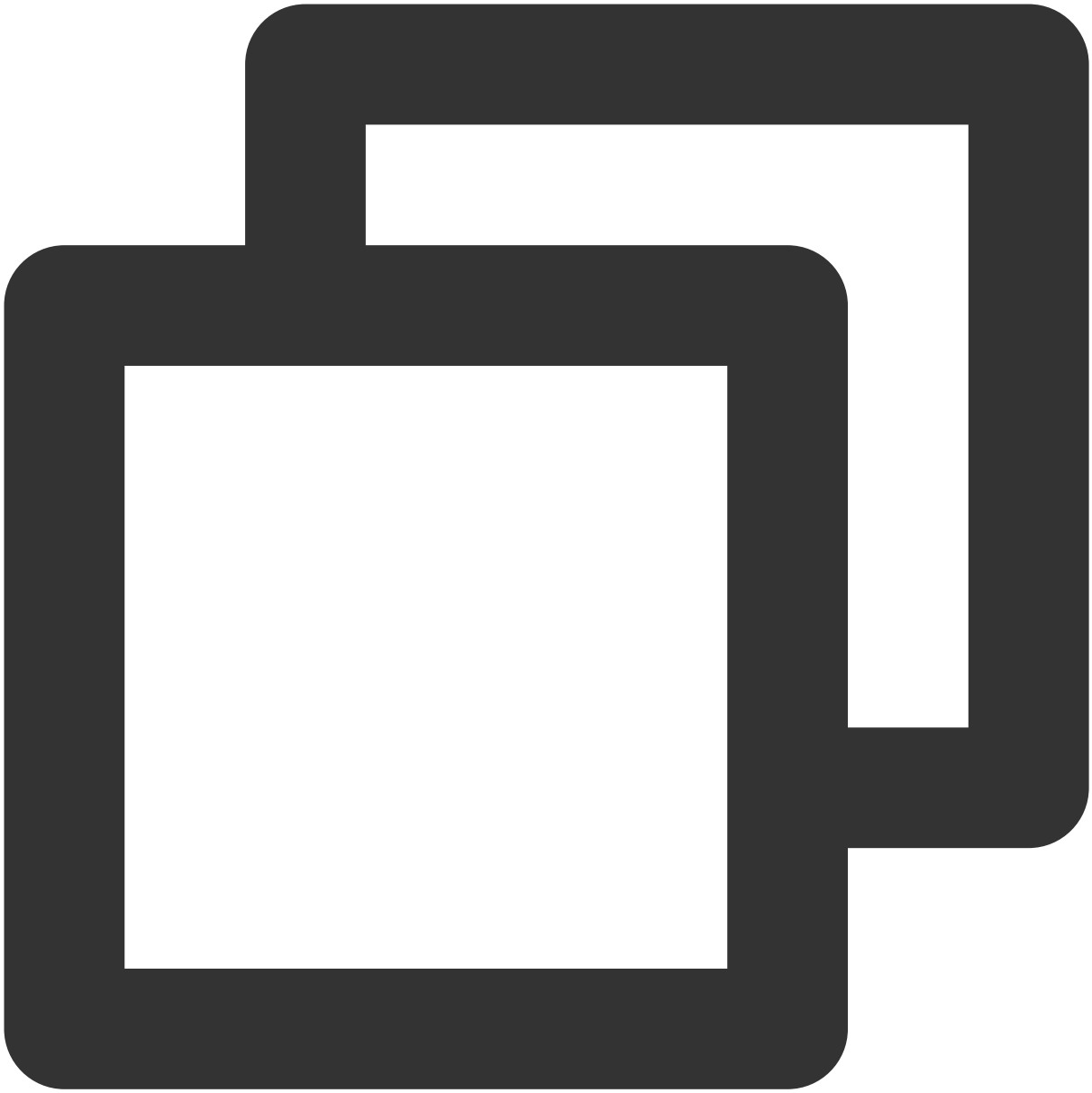
2. To compile the Linux environment for TOA, you need to install the GCC compiler, Make tool, and kernel development package first.

Installation procedure on CentOS



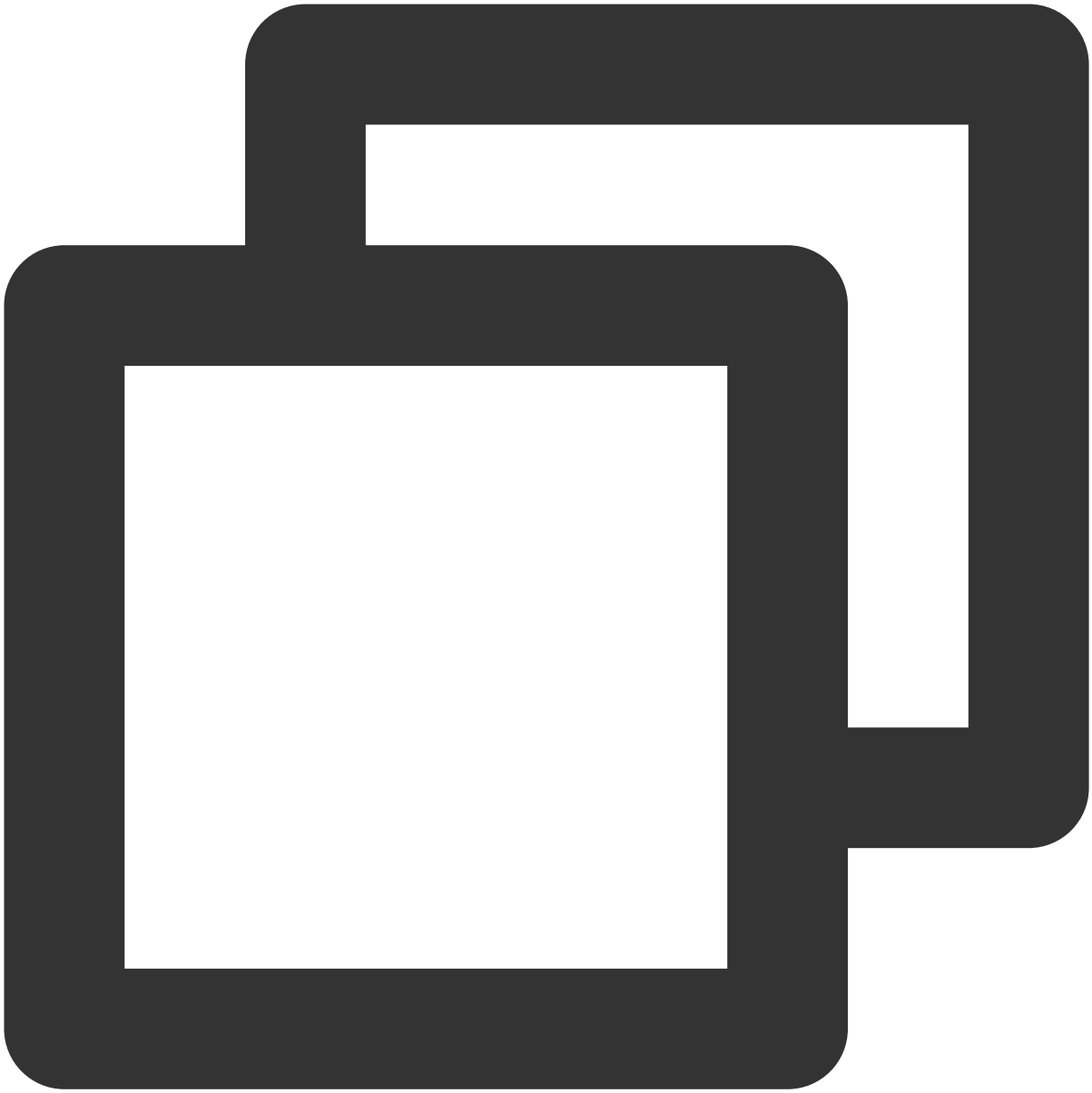
```
yum install gcc
yum install make
//Install the kernel development package. The version of the package header file an
yum install kernel-devel-`uname -r`
yum install devtoolset-8
```

Installation procedure on Ubuntu and Debian



```
apt-get install gcc
apt-get install make
//Install the kernel development package. The version of the package header file an
apt-get install linux-headers-`uname -r`
apt-get install devtoolset-8
```

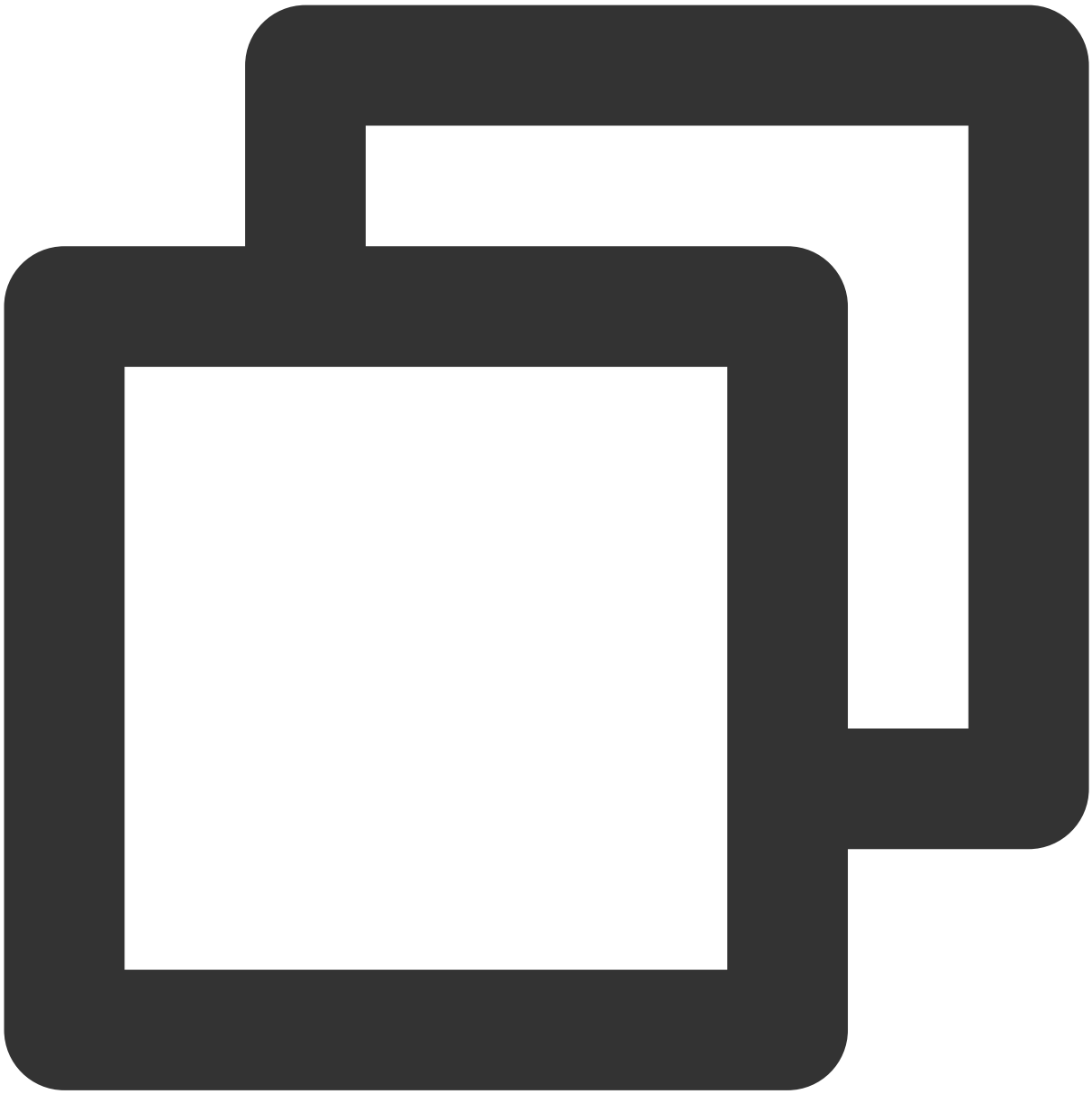
Installation procedure on SUSE



```
zypper install gcc
zypper install make
//Install the kernel development package. The version of the package header file an
zypper install kernel-default-devel
zypper install devtoolset-8
```

3. Change the PATH environment variable to `PATH=/opt/rh/devtoolset-8/root/bin:$PATH`. Before compiling, make sure that the Kernel version matches the GCC version. You can run `dmesg | grep 'Linux version'` to check the Kernel version.

4. Compile the source package to generate the `toa.ko` file. If `warning` and `error` are not prompted during the compilation process, the compilation is successful. Take the source package for Linux OS as an example:



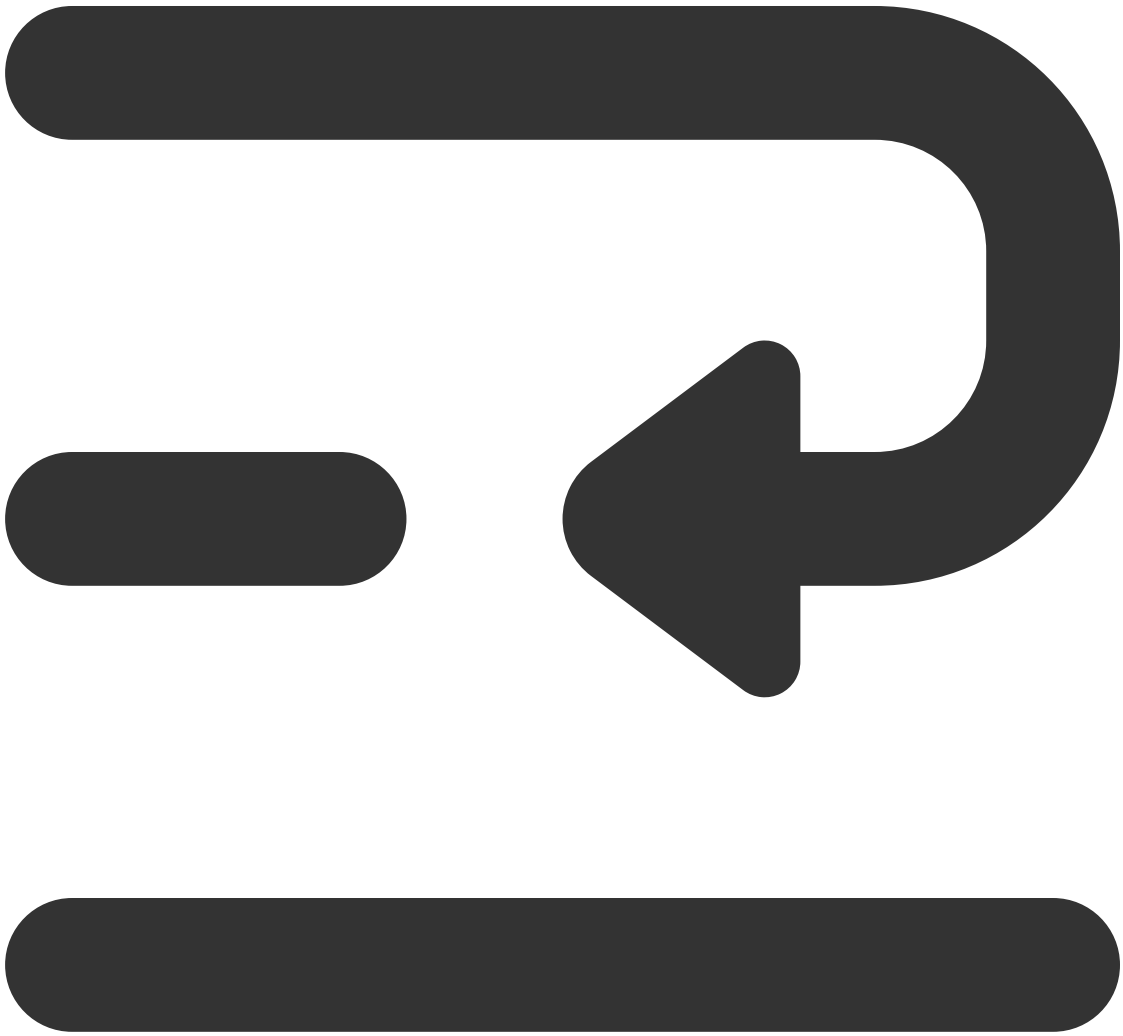
```
tar zxvf tgw_toa_linux_ver.tar.gz
cd tgw_toa_linux_ver//Enter the decompressed directory tgw_toa
make
```

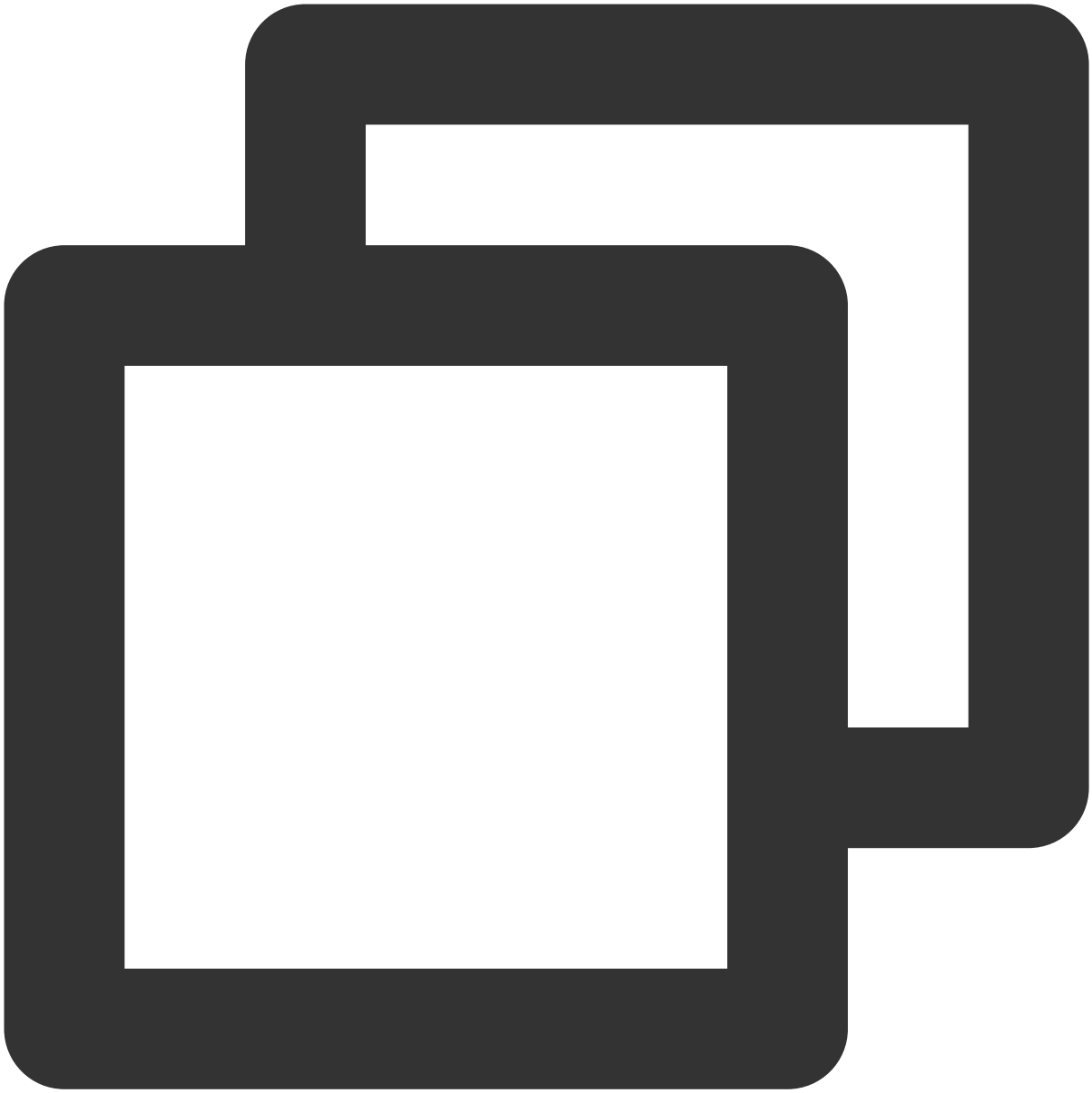
5. After the compilation is successful, perform step 2 to load TOA.

Adapting the Real Sever

Hybrid cloud deployment

When adapting the real server in a hybrid cloud, you only need to call the standard Linux network programming API to obtain the real client IP address without code changes. The following shows a code sample.



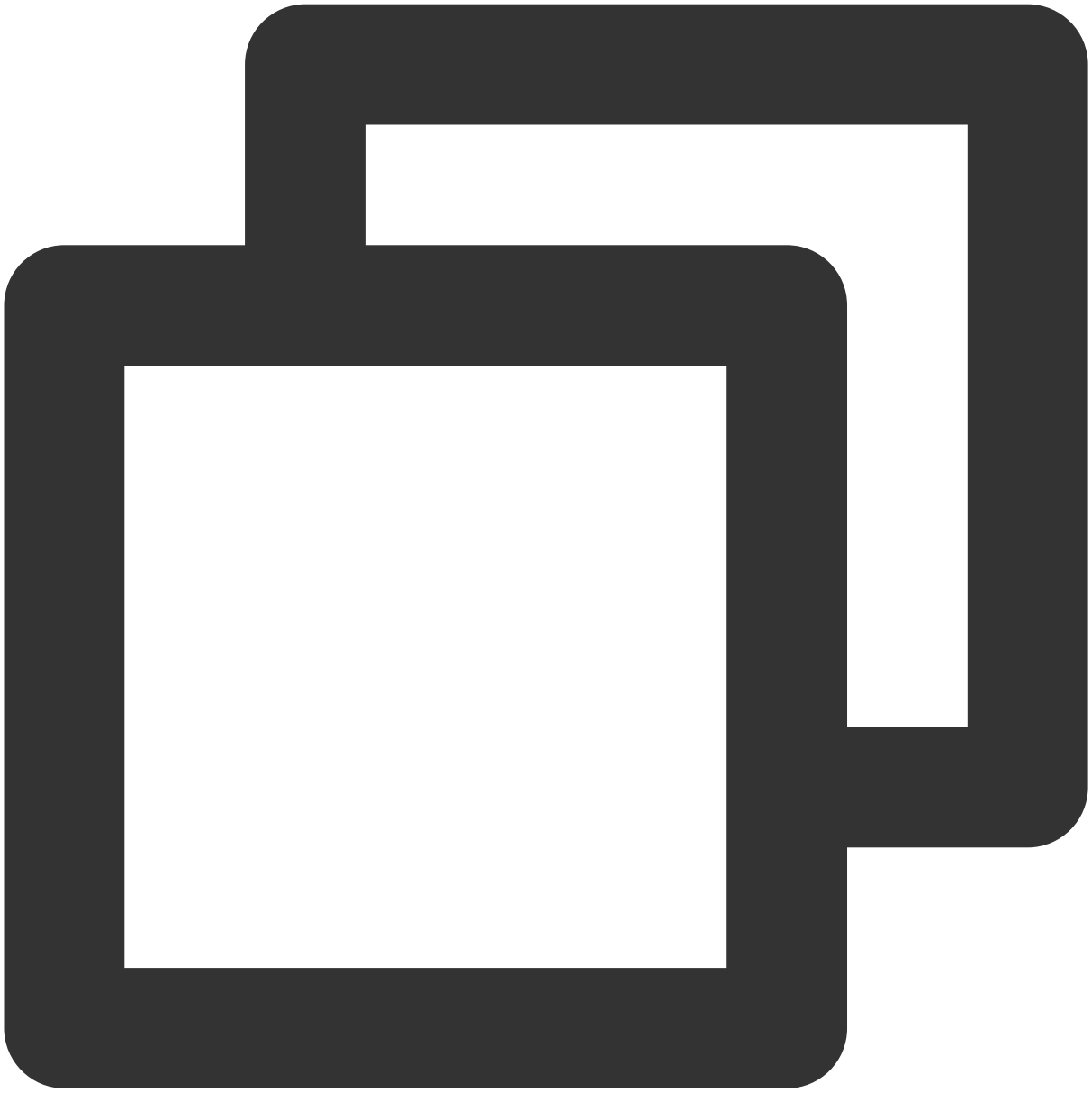


```
struct sockaddr v4addr;
len = sizeof(struct sockaddr);
//`get_peer_name` is the standard Linux network programming API.
if (get_peer_name(client_fd, &v4addr, &len) == 0) {
    inet_ntop(AF_INET, &(((struct sockaddr_in *)&v4addr)->sin_addr), from, sizeof(f
    printf("real client v4 [%s]:%d\\n", from, ntohs(((struct sockaddr_in *)&v4addr)
}
```

NAT64 CLB

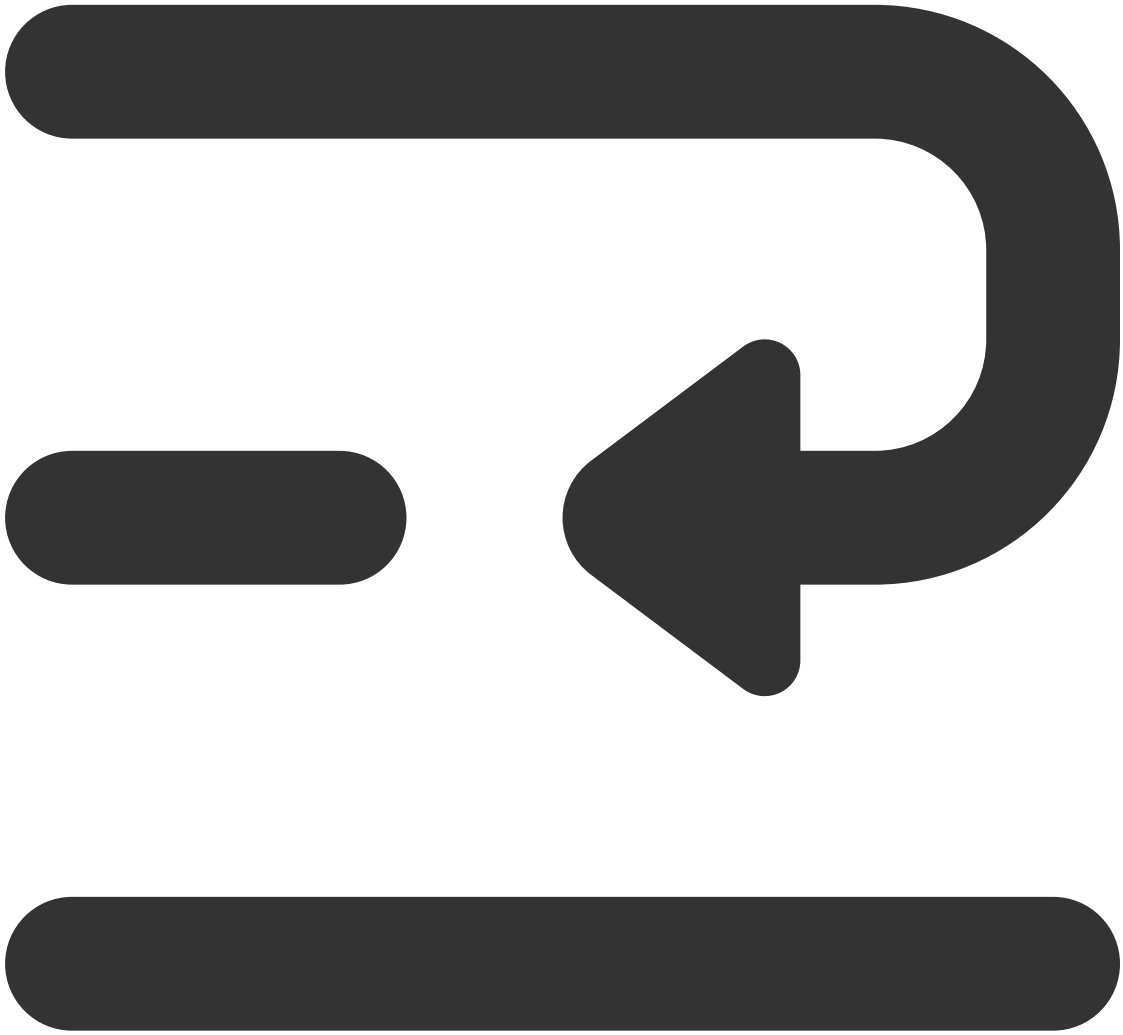
To obtain the real client IP address in NAT64 CLB scenarios, you need to modify the source code after the `toa.ko` kernel module is inserted into the real server and TOA will pass the IP address to the real server.

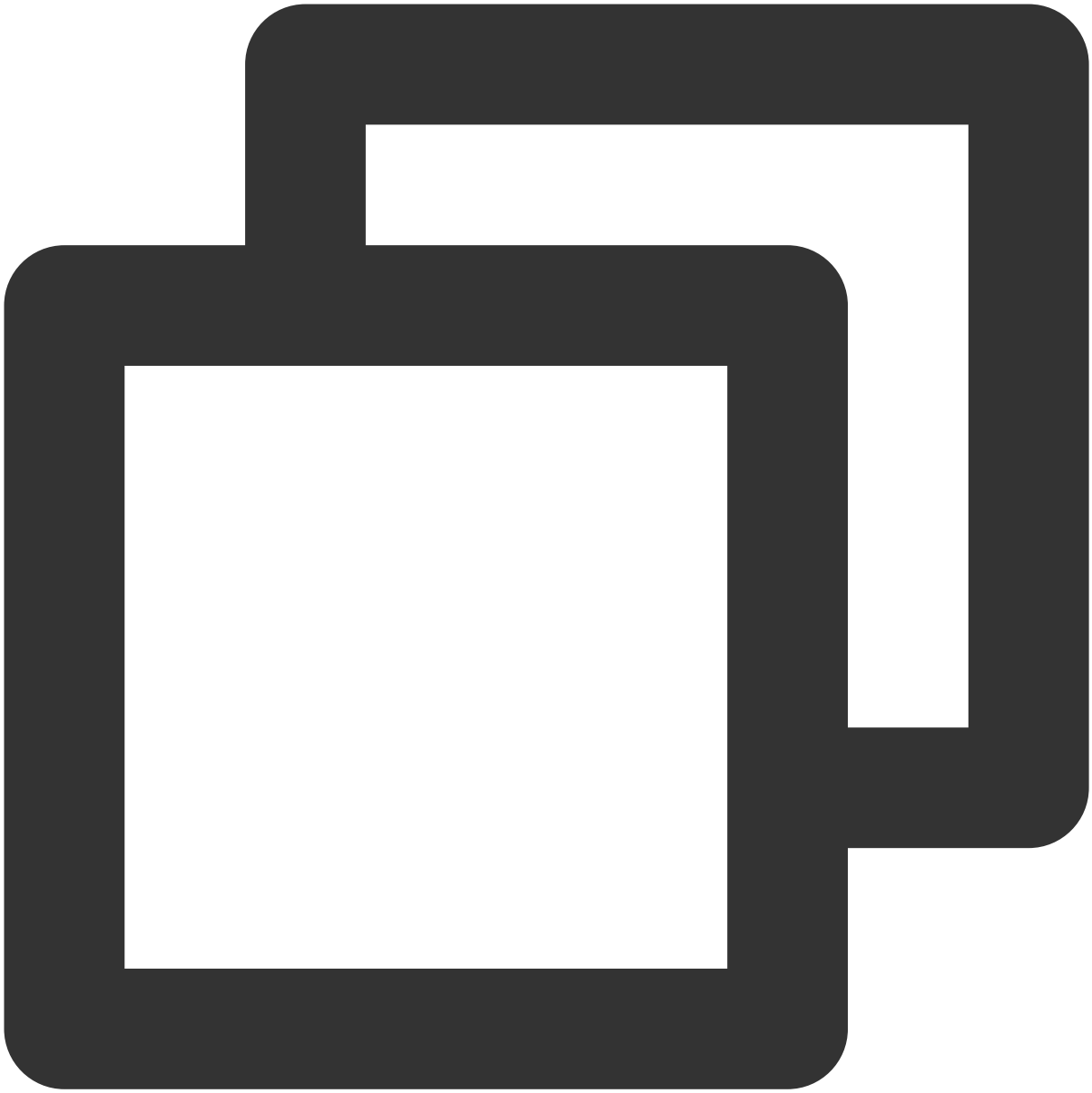
1. Define a data structure to store the IP address.



```
struct toa_nat64_peer {
    struct in6_addr saddr;
    uint16_t sport;
};
....
struct toa_nat64_peer client_addr;
....
```

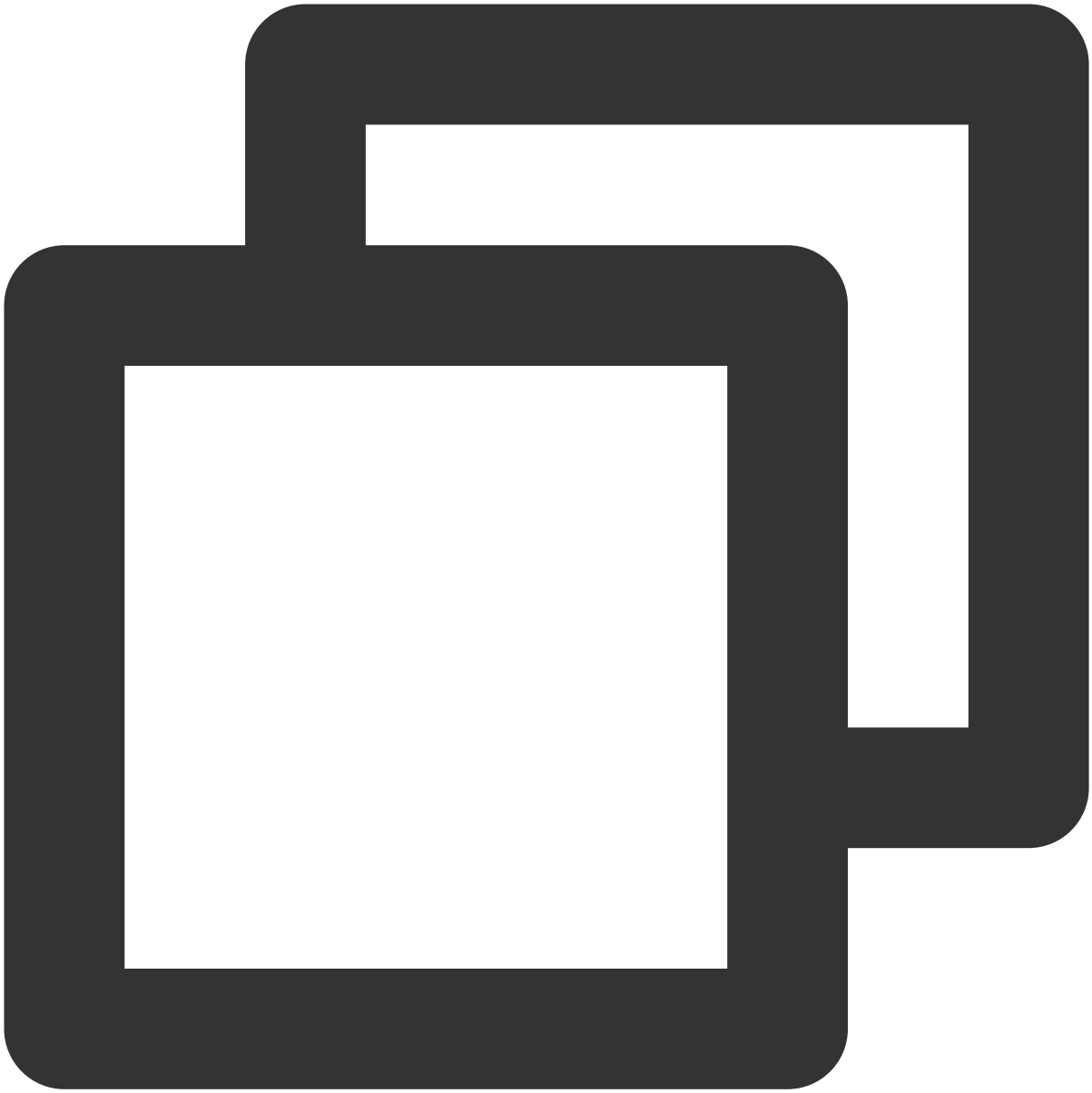
2. Define TOA variables and make calls to obtain the real client IPv6 address.





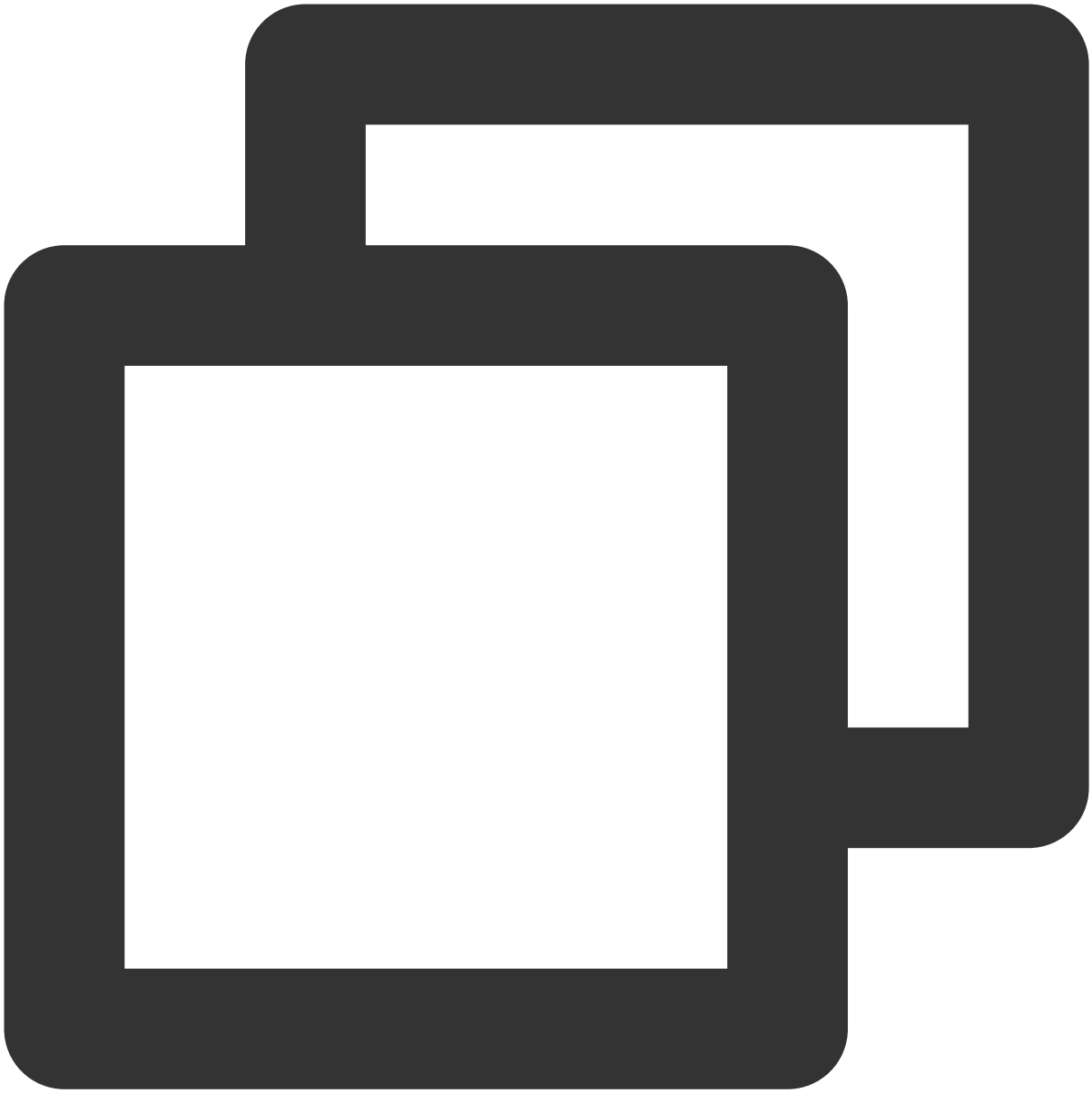
```
enum {
    TOA_BASE_CTL            = 4096,
    TOA_SO_SET_MAX          = TOA_BASE_CTL,
    TOA_SO_GET_LOOKUP       = TOA_BASE_CTL,
    TOA_SO_GET_MAX          = TOA_SO_GET_LOOKUP,
};
getsockopt(client_fd, IPPROTO_IP, TOA_SO_GET_LOOKUP, &client_addr, &len);
```

3. Obtain the real client IP address.



```
real_ipv6_saddr = client_addr.saddr;
real_ipv6_sport = client_addr.sport;
```

A complete configuration sample is as follows:



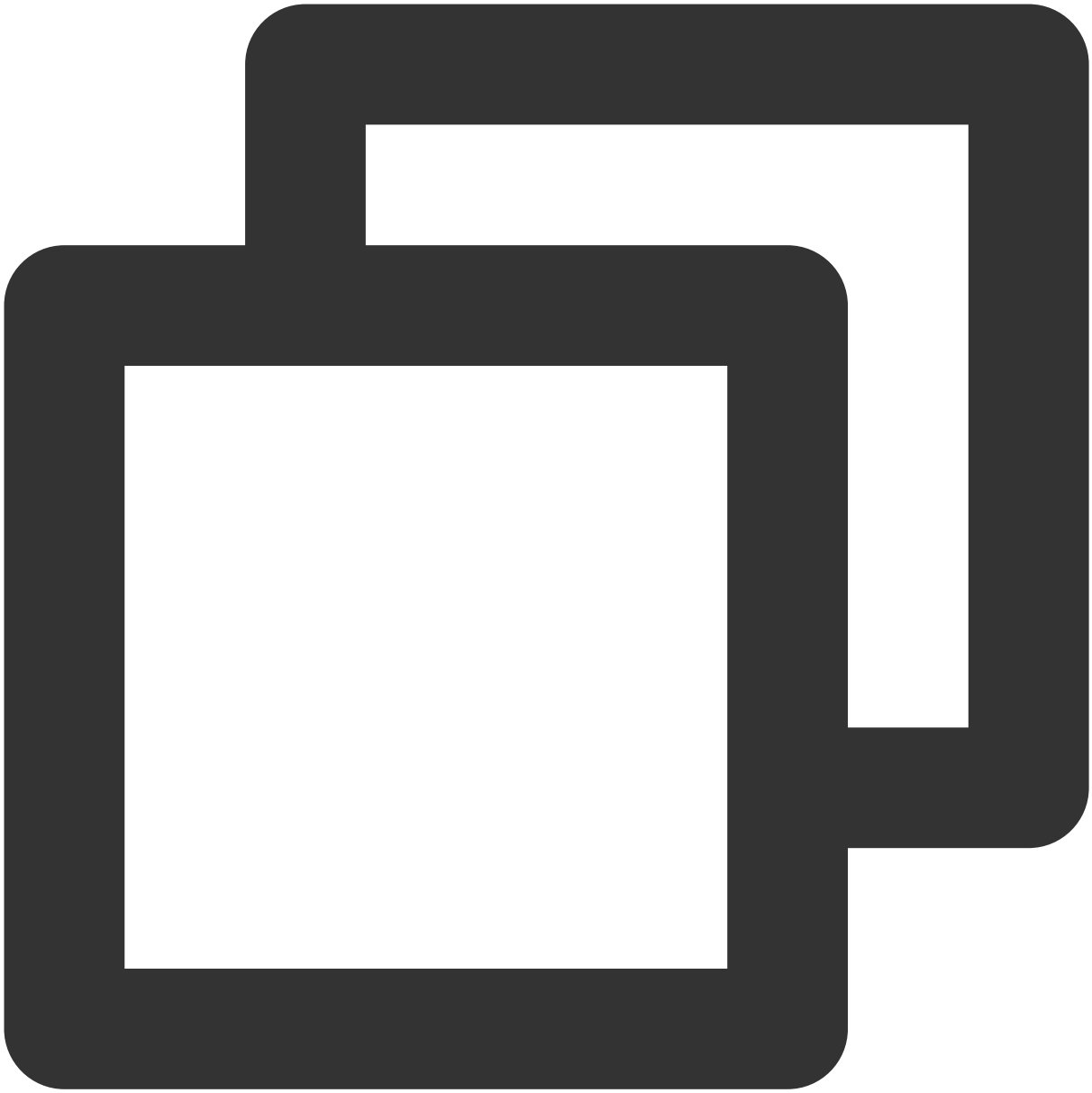
```
//Define TOA variables. Set `TOA_BASE_CTL` to 4096.
enum {
    TOA_BASE_CTL            = 4096,
    TOA_SO_SET_MAX          = TOA_BASE_CTL,
    TOA_SO_GET_LOOKUP       = TOA_BASE_CTL,
    TOA_SO_GET_MAX          = TOA_SO_GET_LOOKUP,
};
//Define a data structure to store the IP address.
struct toa_nat64_peer {
    struct in6_addr saddr;
    uint16_t sport;
```

```
};
//Declare the variable that is used to store the address.
struct toa_nat64_peer client_addr;
.……
//Get the file descriptor of the client, where `listenfd` is the listening file des
client_fd = accept(listenfd, (struct sockaddr*)&caddr, &length);
//Make calls to get the real client IP address of the user in the NAT64 scenario.
char from[40];
int len = sizeof(struct toa_nat64_peer);
if (getsockopt(client_fd, IPPROTO_IP, TOA_SO_GET_LOOKUP, &client_addr, &len) == 0)
    inet_ntop(AF_INET6, &client_addr.saddr, from, sizeof(from));
    //Obtain the source IP and source port
    printf("real client [%s]:%d\\n", from, ntohs(client_addr.sport));
}
```

NAT64 CLB used in a hybrid cloud

To obtain the real client IP address in the scenario where NAT64 CLB is used in a hybrid cloud, you need to modify the source code after the `toa.ko` kernel module is inserted into the real server and TOA will pass the IP address to the real server.

A complete configuration sample is as follows:



```
//Define TOA variables. Set `TOA_BASE_CTL` to 4096.
enum {
    TOA_BASE_CTL = 4096,
    TOA_SO_SET_MAX = TOA_BASE_CTL,
    TOA_SO_GET_LOOKUP = TOA_BASE_CTL,
    TOA_SO_GET_MAX = TOA_SO_GET_LOOKUP,
};

//Define a data structure to store the IP address.
struct toa_nat64_peer {
    struct in6_addr saddr;
```

```
    uint16_t sport;
};

//Declare the variable that is used to store the address.
struct toa_nat64_peer client_addr_nat64;
.......
//Get the file descriptor of the client, where `listenfd` is the listening file des
//Make calls to get the real client IP in the NAT64 scenario.
char from[40];
int len = sizeof(struct toa_nat64_peer);
int ret;
ret = getsockopt(client_fd, IPPROTO_IP, TOA_SO_GET_LOOKUP, &client_addr_nat64, &len
if (ret == 0) {
    inet_ntop(AF_INET6, &(client_addr_nat64.saddr), from, sizeof(from));
    //Obtain the source IP and source port.
    printf("real client v6 [%s]:%d\\n", from, ntohs(client_addr_nat64.sport));
} else if (ret != 0) {
    struct sockaddr v4addr;
    len = sizeof(struct sockaddr);
    //Obtain the source IP and source port:
    //In the hybrid cloud deployment scenario, the source IP address is the IP addr
    //In the non-hybrid cloud deployment scenario, the source IP address is the cli
    //The semantics of this function is to get the real client address and port.
    if (get_peer_name(client_fd, &v4addr, &len) == 0) {
        inet_ntop(AF_INET, &(((struct sockaddr_in *)&v4addr)->sin_addr), from, size
        printf("real client v4 [%s]:%d\\n", from, ntohs(((struct sockaddr_in *)&v4a
    }
}
```

(Optional) Monitoring TOA Status

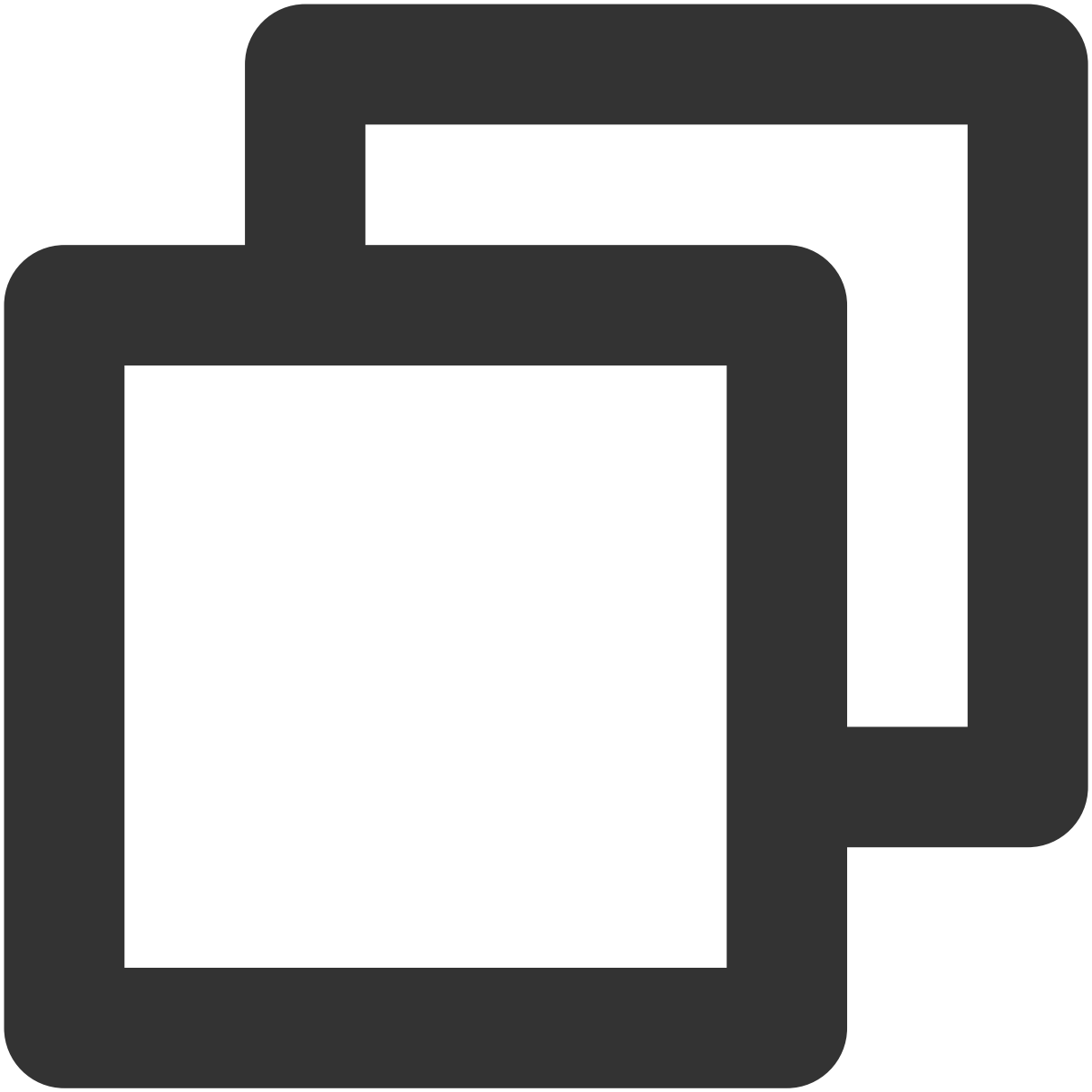
To ensure execution stability, this kernel module allows you to monitor status. After inserting the `toa.ko` kernel module, you can monitor the TOA working status on the host of the container in either of the following ways.

Method 1: Checking the IPv6 address stored in TOA

Run the following command to check the IPv6 address stored in TOA:

Note:

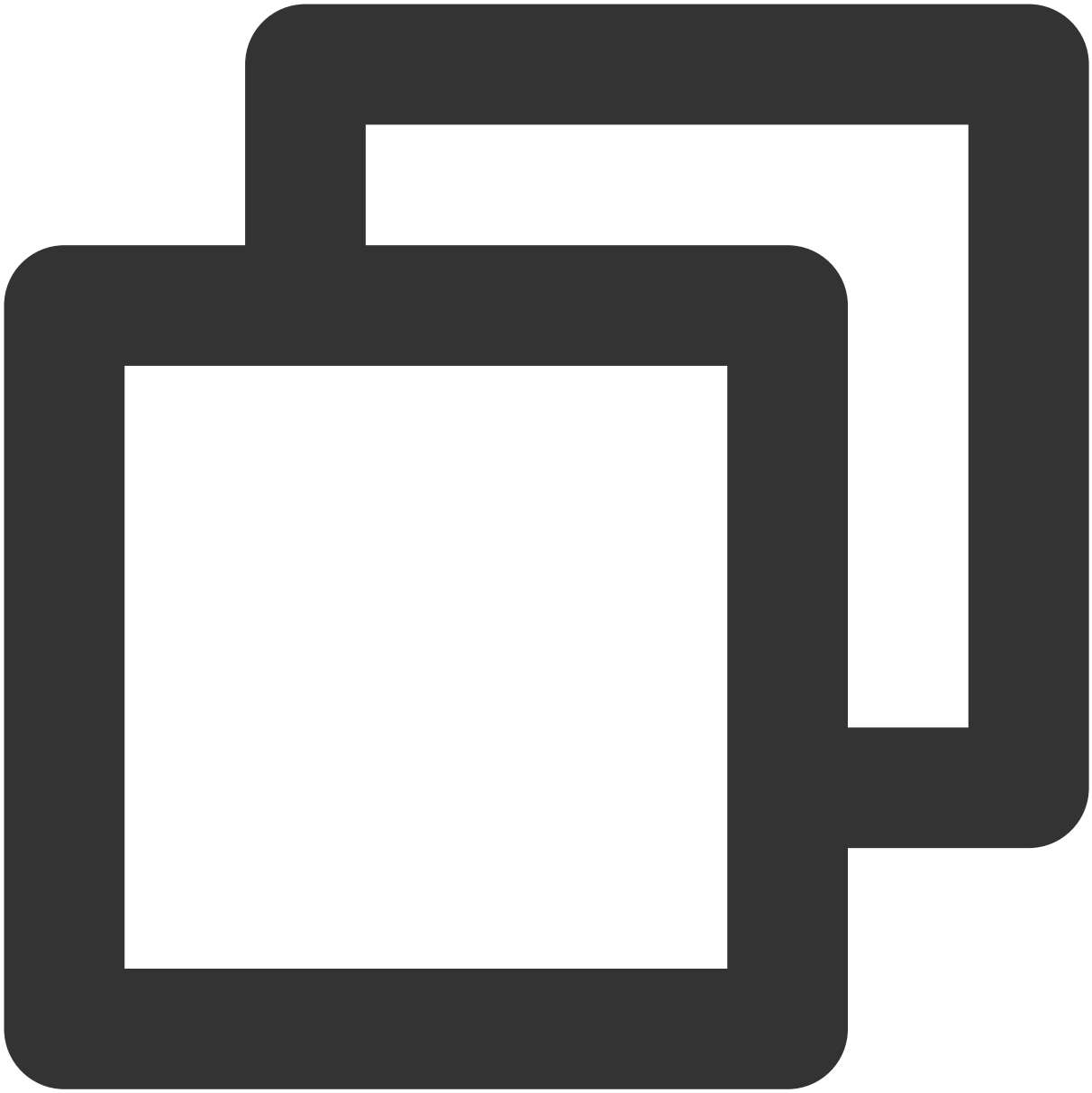
Executing this command may degrade performance. Proceed with caution.



```
cat /proc/net/toa_table
```

Method 2: Checking TOA metrics

Run the following command to check TOA metrics:



```
cat /proc/net/toa_stats
```

The monitoring metrics are described as follows:

Metric	Description
syn_recv_sock_toa	Receives connections with TOA information.
syn_recv_sock_no_toa	Receives connections without TOA information.
getname_toa_ok	This count increases when you call `getsockopt` and obtain the source IP

	address successfully or when you call `accept` to receive client requests.
getname_toa_mismatch	This count increases when you call `getsockopt` and obtain a source IP address that does not match the required type. For example, if a client connection contains a source IPv4 address whereas you obtain an IPv6 address, the count will increase.
getname_toa_empty	This count increases when the `getsockopt` function is called in a client file descriptor that does not contain TOA.
ip6_address_alloc	Allocates space to store the information when TOA obtains the source IP address and source port saved in the TCP data packet.
ip6_address_free	When the connection is released, TOA will release the memory previously used to save the source IP and source port. If all connections are closed, the total count of `ip6_address_alloc` for each CPU should be equal to the count of this metric.

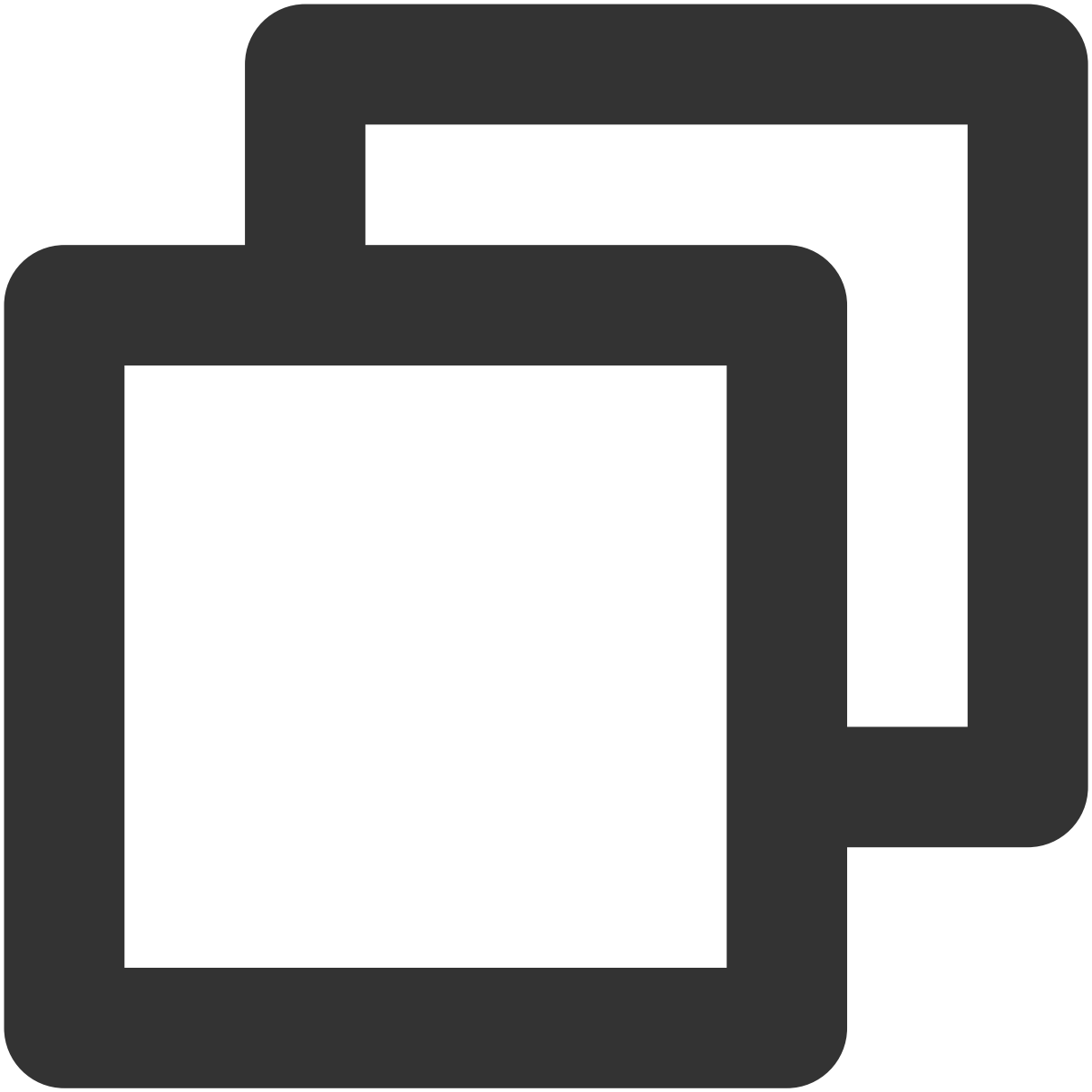
FAQs

Why do I need to modify the server program after TOA is inserted for NAT64 CLB?

This is because that the client IPv4 address is converted to an IPv6 address in the hybrid cloud deployment scenario, which is different from the NAT64 CLB scenario where the client IP type remains unchanged. Therefore, you need to modify the server program so that the server can understand the IPv6 address.

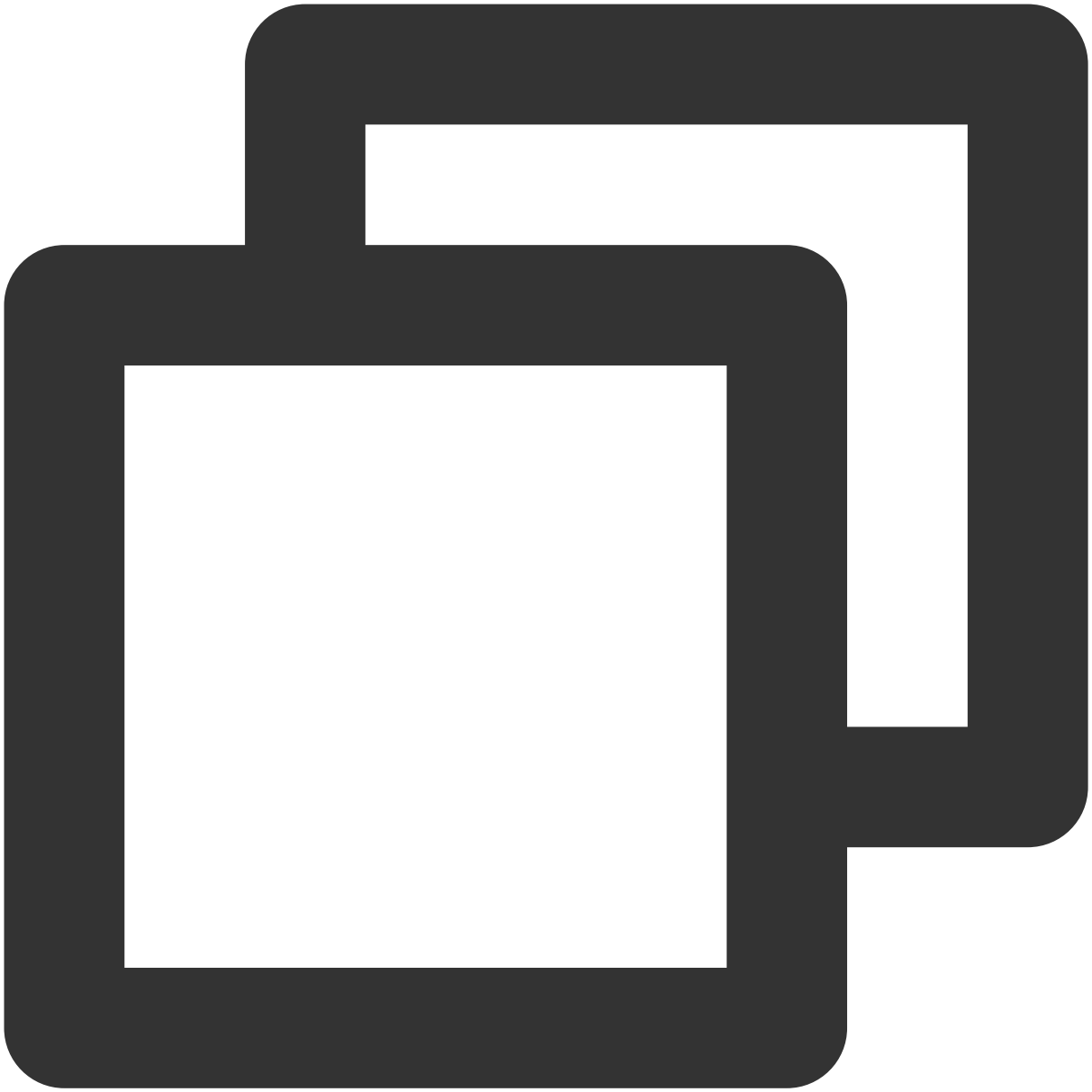
How do I know my OS is based on the Linux distribution or TLinux kernel?

Run the following command to check your kernel version. If you see `tlinux` in the command output, you are using TLinux OS, while `linux` indicates that you are using Linux OS.



```
uname -a
```

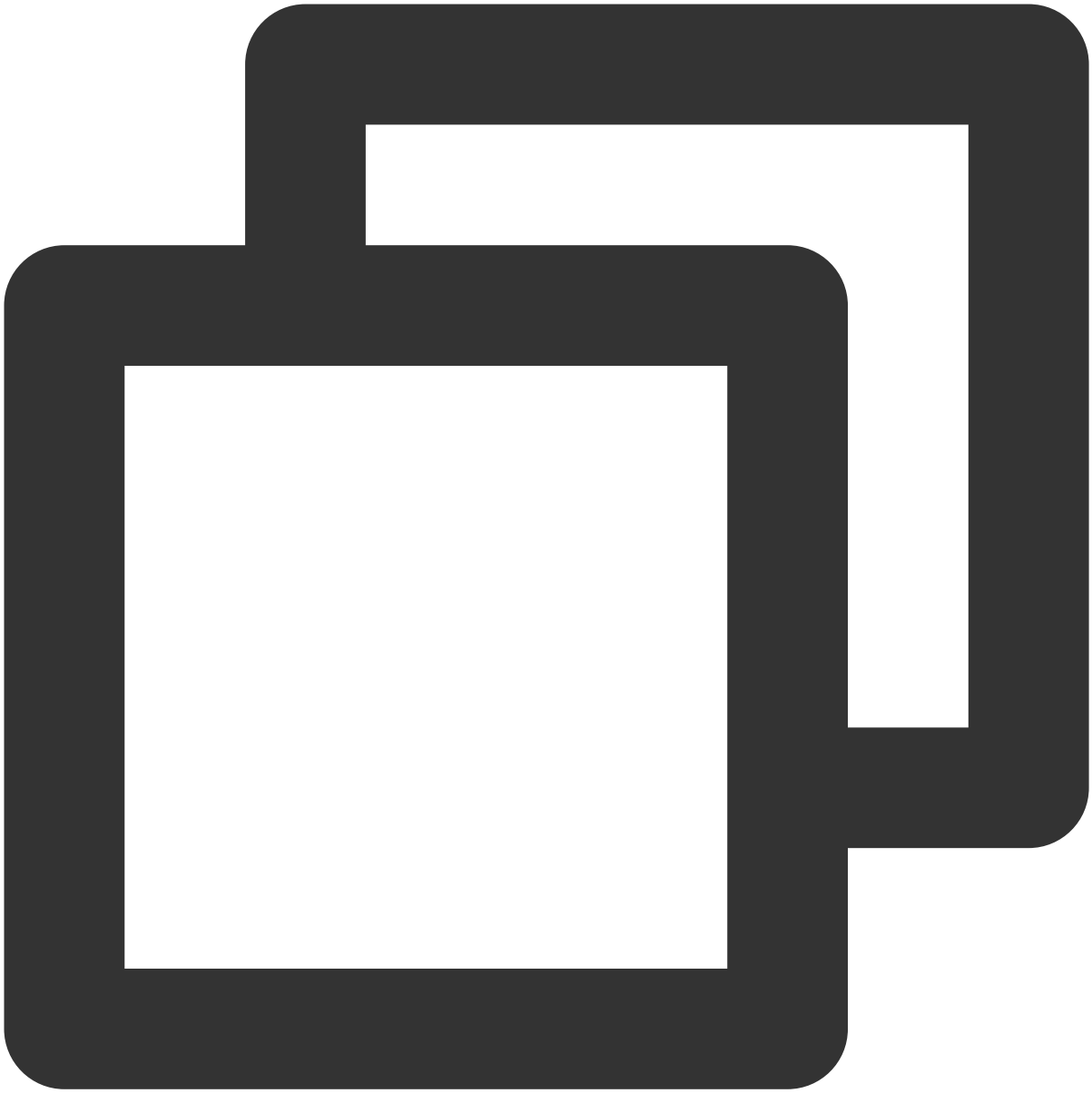
You can also check the version by running the following command. If `tlinux` or `tl2` is returned, you are using TLinux OS.



```
rpm -qa | grep kernel
```

How do I perform preliminary checks when I failed to obtain the source IP address?

1. Run the following command to check whether TOA has been loaded:



```
lsmod | grep toa
```

2. Check whether the server program has made correct calls to obtain the source IP address. You can refer to Adapting the Real Sever.

3. Capture TCP packets on the server and check whether the packets contain the source IP information.

If `unknown-200` is displayed in the `tcp option` output, the real client IP address after SNAT is inserted into the field `tcp option` .

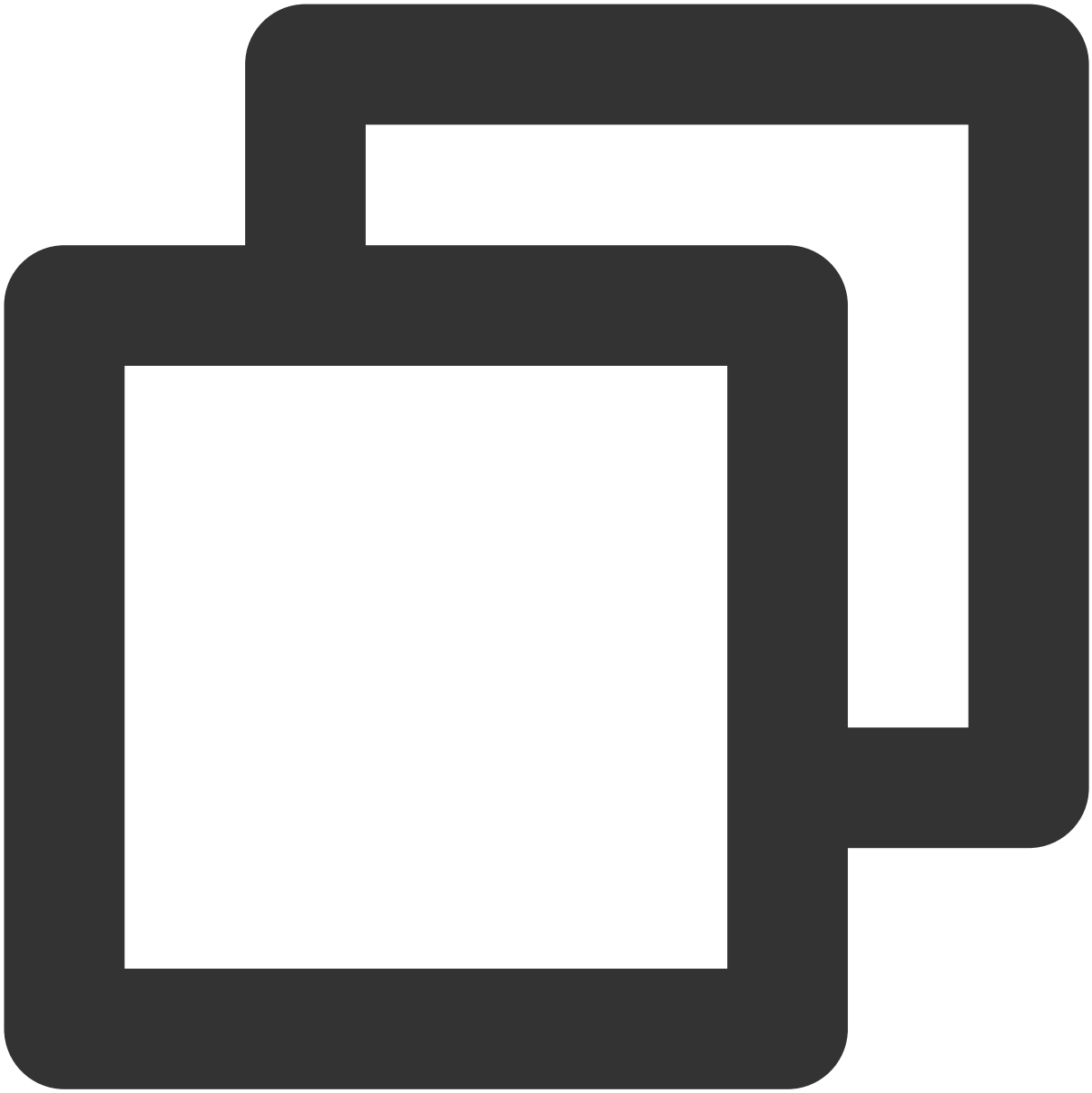
If `unknown-253` is displayed, the real client IPv6 address is inserted in the NAT64 scenario.

```
[root@VM-0-133-centos ]# tcpdump -i any "ip[40:1]==200" -c 100
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
18:04:24.864649 IP 192.168.0.177.23638 > VM-0-133-centos.webcache: Flags [.], ack 3309146461, win 229, options [unknown-200 0x
2662ac13bca,nop,nop,TS val 2243901958 ecr 3395797654], length 0
18:04:24.864679 IP 192.168.0.177.23638 > VM-0-133-centos.webcache: Flags [P.], seq 0:154, ack 1, win 229, options [unknown-200
0xa2662ac13bca,nop,nop,TS val 2243901958 ecr 3395797654], length 154: HTTP: GET /data/1K HTTP/1.1
```

4. If the packet containing the TOA address has been sent to the server, compile `toa.ko` to a DEBUG version, and further locate the problem through kernel logs. In the downloaded TOA source directory, add the DEBUG compilation option to the make file.

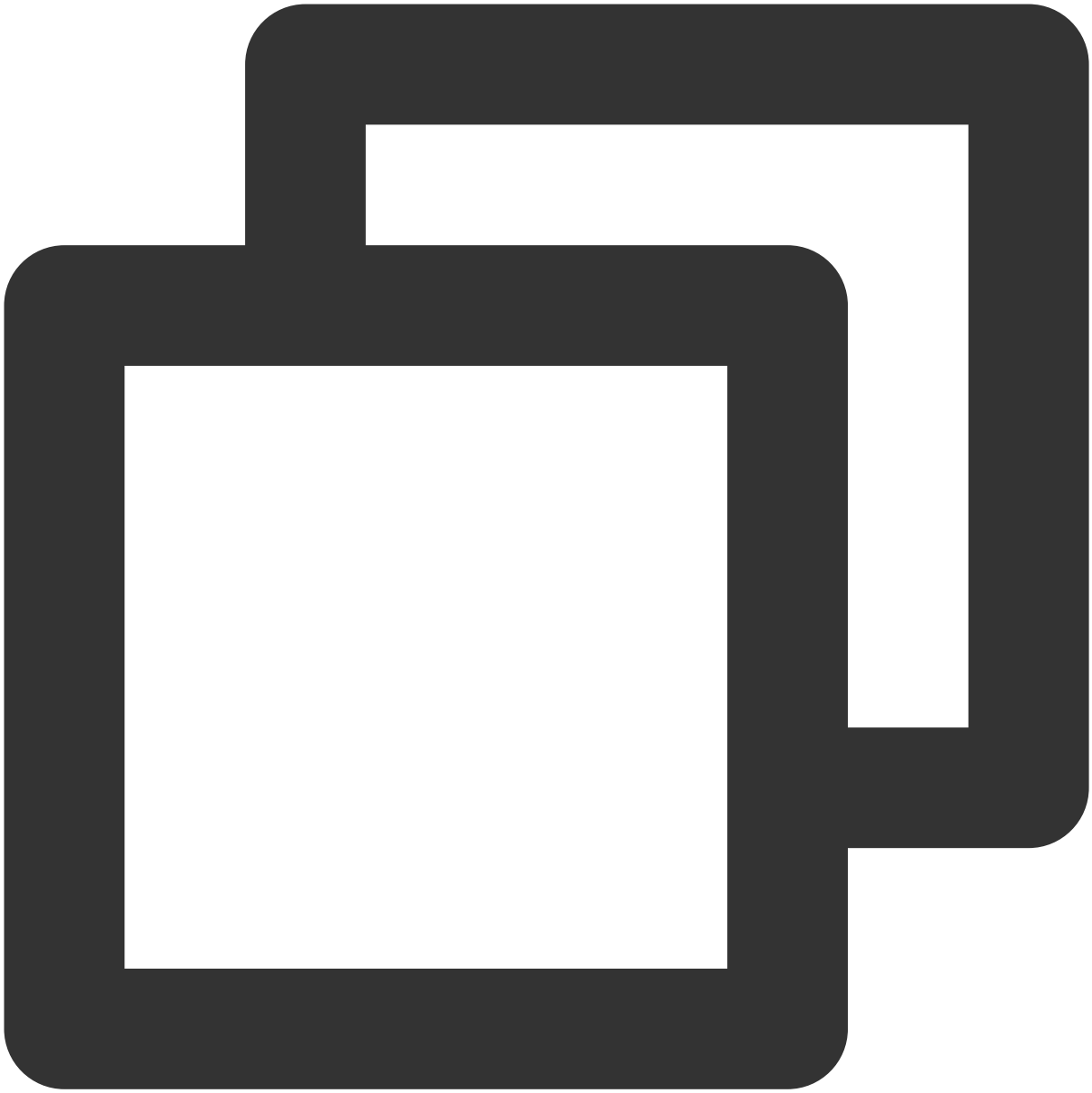
```
endif
PWD       := $(shell pwd)

ccflags-y += -DTOA_NAT64_ENABLE -DTOA_DEBUG_ENABLE

ifeq ($(DEBUG), 1)
ccflags-y += -g -O0
endif
```

5. Run the following commands to compile again:



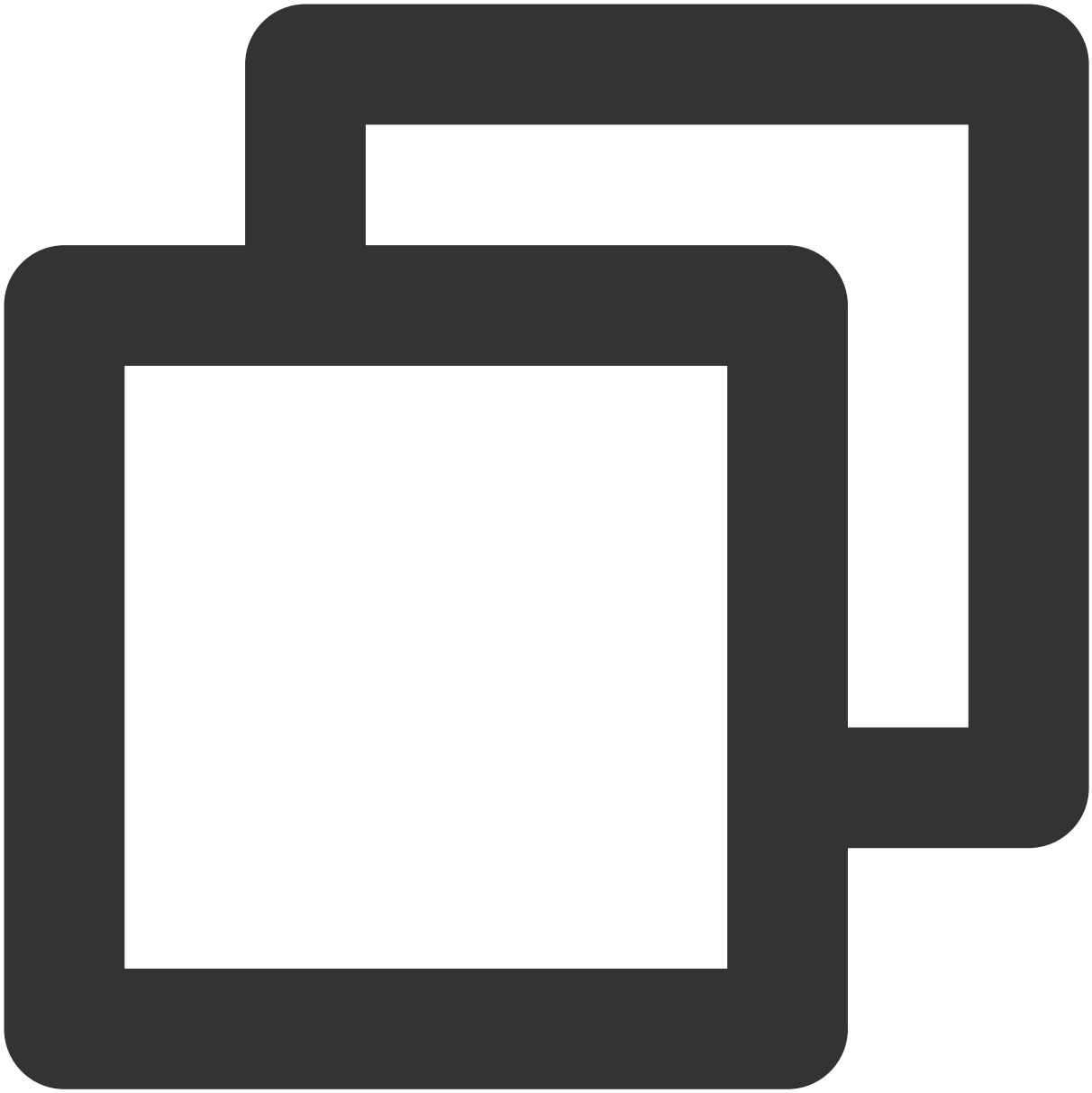
```
make clean
make
```

6. Run the following commands to uninstall the original `toa.ko` file and install the latest one.



```
rmmod toa
insmod ./toa.ko
```

7. Run the following command to observe kernel logs.



```
dmesg -Tw
```

If you see the following message, TOA is working normally. You can further check whether the server program has made calls to obtain the real client IP address, or whether the API is used incorrectly.

```
[Wed Dec 29 18:07:11 2021] [DEBUG] TOA: inet_getname_toa called, sk->sk_user_data is 000000003088927f
[Wed Dec 29 18:07:11 2021] [DEBUG] TOA: inet_getname_toa: set new sockaddr, ip 192.168.0.177 -> 42.193.59.202, p
618
```

8. If you failed to find out the problem with the preceding steps, please submit a ticket.

Best Practices for Configuring Load Balancing Monitoring Alerts

Last updated：2024-01-04 14:39:00

To enhance the monitoring system of Cloud Load Balancer (CLB), we have integrated the data collection and alarm capabilities of Tencent Cloud Observability Platform (TCOP) to create a unified early warning mechanism. This platform allows you to comprehensive monitor the resource usage, performance, and operational status of CLB, configure monitoring alarms for the required instances, and set alarm triggering rules for monitoring metrics and events. In the event of an exception in the monitoring metrics of the instances, you will receive an exception alarm notification promptly for you to respond and address the issue in a timely manner.

Application Scenario

You can create metric alarms for required instances, enabling CLB instances to promptly send alarm notifications to the concerned users when operational conditions reach a certain threshold. In this way, users can address potential emergencies more conveniently, thereby enhancing operational efficiency and reducing operational costs.
This document uses the standard type of public network CLB instances as examples to describe the process of configuring alarms for public network CLB instances that have been upgraded to the LCU-supported type. For LCU-supported type specifications, see LCU-supported Type Specifications.

Prerequisites

You have successfully created a CLB instance and configured the listener. For more information, see Getting Started.
You have successfully bound the real servers. For more information, see Binding Real Servers.
According to this instance, the target instance must have been upgraded to the LCU-supported type. For details, see Upgrade to a LCU-supported instances.

Concepts

Terminology	Definition
Alarm Policy	Comprised of policy name, policy type, alarm object, trigger condition, and notification template.
Policy Type	The alarm policy type serves to categorize policies and align them with specific cloud products. For example, selecting the CVM policy allows you to customize metric alarms for CPU usage,

	disk usage, and so on.
Trigger Condition	A trigger condition is a coherent statement made up of a metric, comparison operator, threshold, statistical period, and duration over N monitoring data points.
Monitoring Type	The monitoring types include Cloud Product Monitoring and RUM.
Notification Template	A notification template can be quickly integrated with multiple policies, suitable for various scenarios in which alarm notifications are received. For details, see Creating Notification Template.

Metric Introduction

The core metrics for determining whether an LCU-supported instance has exceeded its limit include ClientConcurConn, ClientNewConn, TotalReq, ClientOuttraffic, and ClientIntraffic. Therefore, it is important to pay attention to the usage alarm metrics of the preceding core metrics, as listed in the following table. The drop/usage monitoring metrics are currently in beta testing. If you want to use them, please submit a ticket application. For more alarm metrics description, see Alarming Metric Descriptions.

Dimension	Alarm Policy Type	Alarm Policy	Alarm Metrics	Metric Description
Instance	Public network CLB instances	Drop/usage monitoring	IntrafficVipRatio	The usage of bandwidth used by the client to access a CLB instance through a public network in the statistical period.
			OuttrafficVipRatio	The usage of bandwidth used by a CLB instance to access the public network in the statistical period.
			ConcurConnVipRatio	The usage that is obtained from comparing the number of concurrent connections from the client to the CLB with the upper limit of concurrent connection count performance of the LCU-supported type specification at a certain moment within the statistical period.
			NewConnVipRatio	The usage that is obtained from comparing the number of new connections from the client to the CLB

				with the upper limit of new connection count performance of the LCU-supported type specification at a certain moment within the statistical period.
		QPS Related Monitoring	QpsVipRatio	The usage that is obtained from comparing the QPS of a CLB to the capacity limit of QPS performance of the performance capacity specification at a specific point within the statistical period.

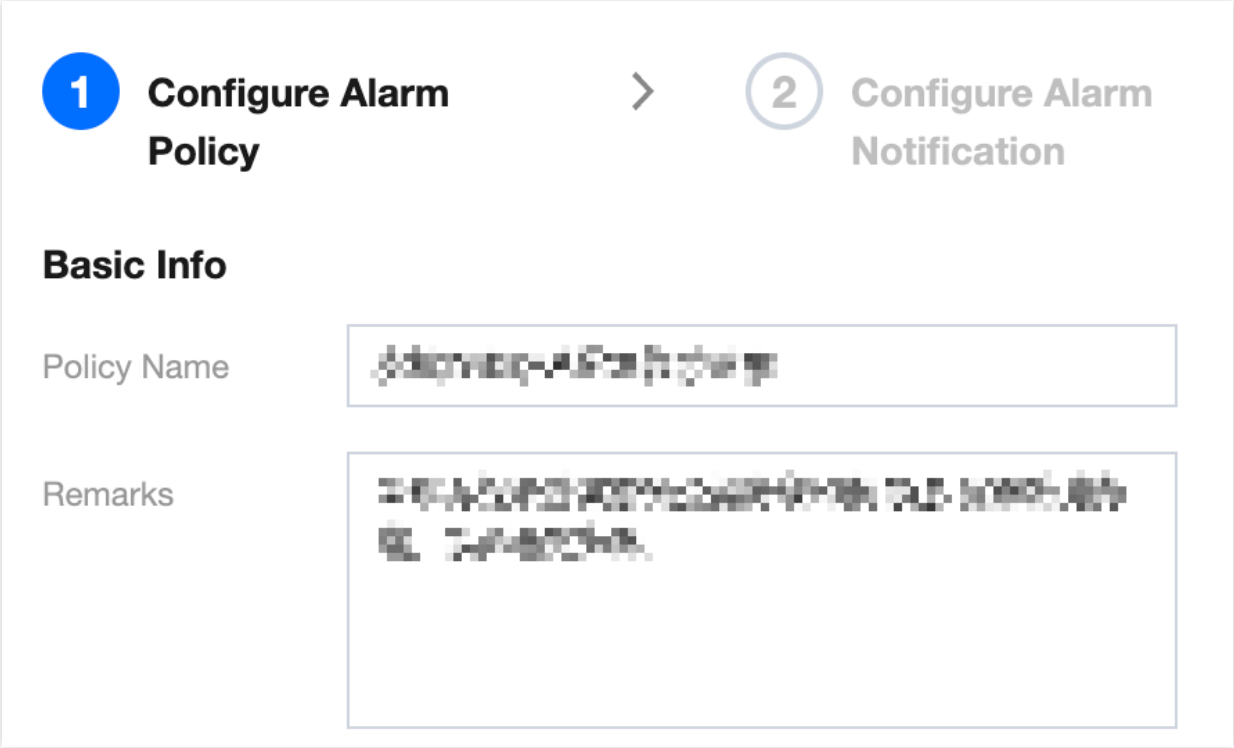
Directions:

1. Log in to the TCOP.
2. On the left sidebar, click **Alarm Management** > **Alarm Configuration** >**Alarm Policy** to access the management page.
3. Click **Create New Policy** and configure the following options.

3.1 Basic Information

Policy Name: Enter a policy name, up to 60 characters in length.

Remarks: Enter any remarks, up to a maximum of 100 characters.



3.2 Configuring Alarm Rule

Monitoring Type: Choose Cloud Product Monitoring.

Policy Type: Choose **CLB** > **Public CLB Instance** > **About drop/usage monitor**.

Project: Select the project to which the policy belongs. The project is for categorization and permission management in alarm policies and is not tightly bound with the project of cloud product instances.

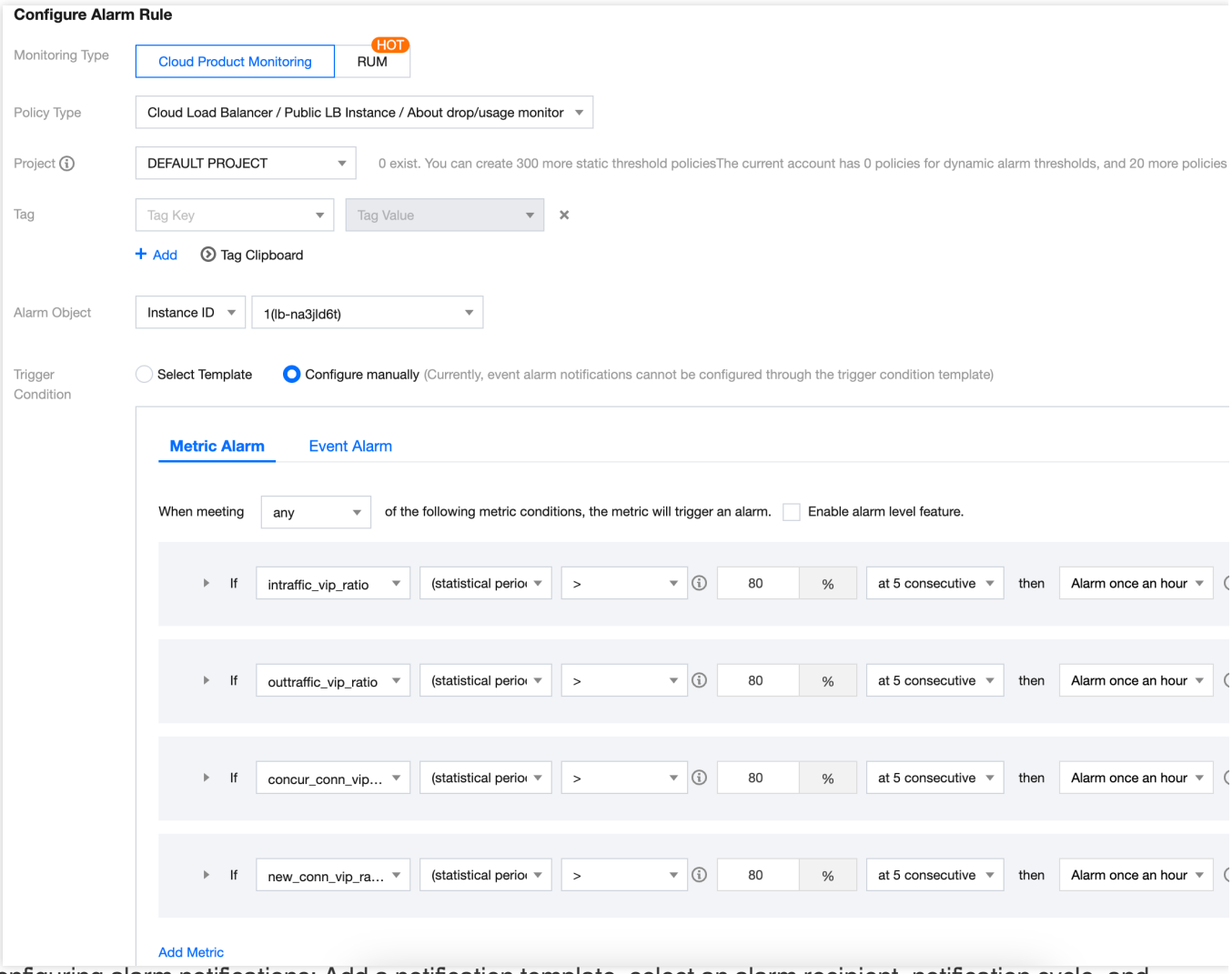
Tag: Select the tag to which the policy belongs.

Alarm Object: Choose the target instance as the alarm object.

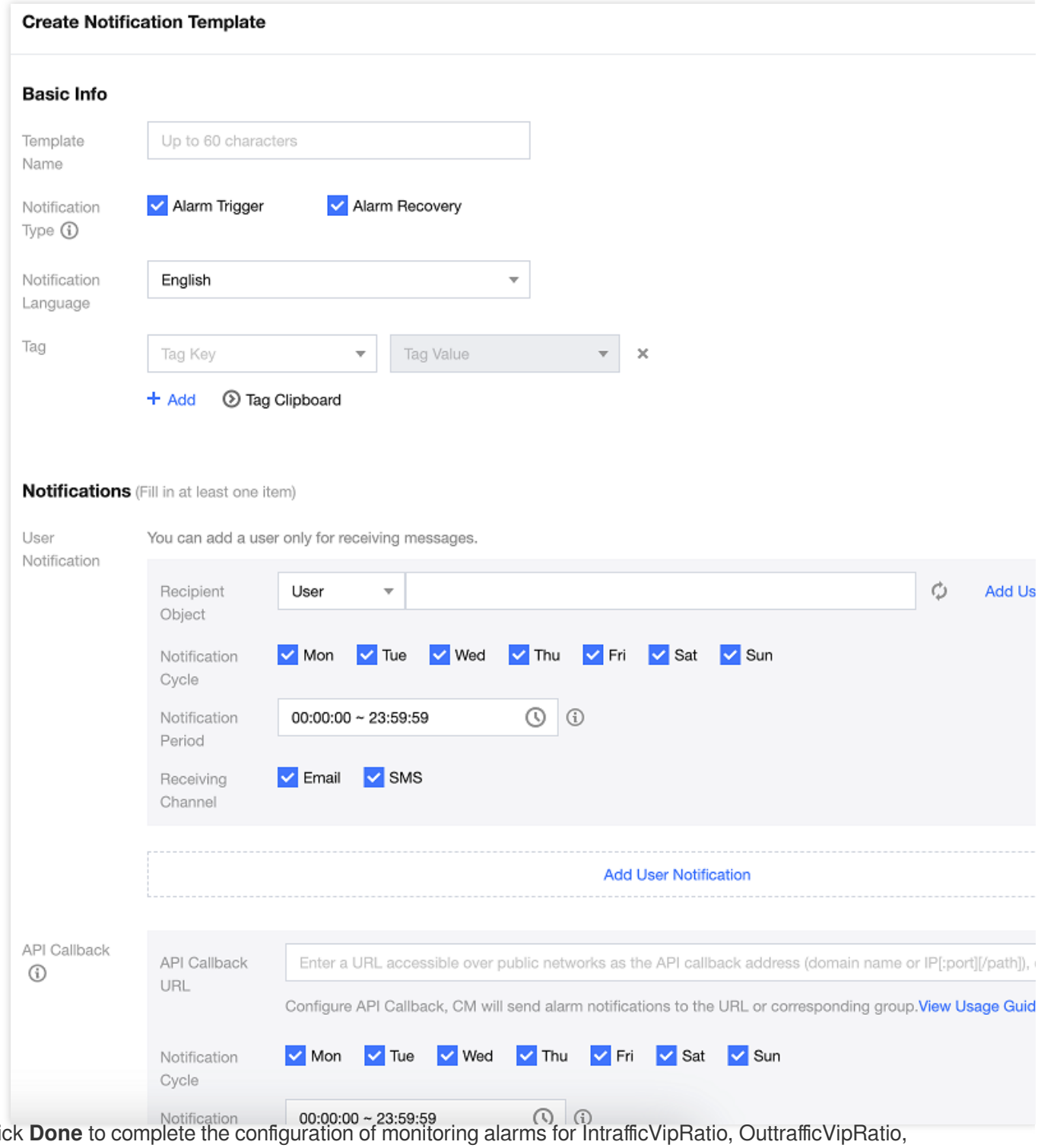
Trigger Condition: A semantic condition consisting of an alarm metric, comparison relationship, threshold, duration for N monitoring data points, and alarm frequency.

For example, if the alarm metric is set to **IntrafficVipRatio**, comparison to **>**, threshold to **80%**, continuous monitoring data points to **5 data points** and alarm frequency is set to **Alarm once an hour**, an alarm will be triggered when the inbound bandwidth usage of a CLB instance is greater than 80% for five consecutive times, with the alarm frequency set to trigger once every hour.

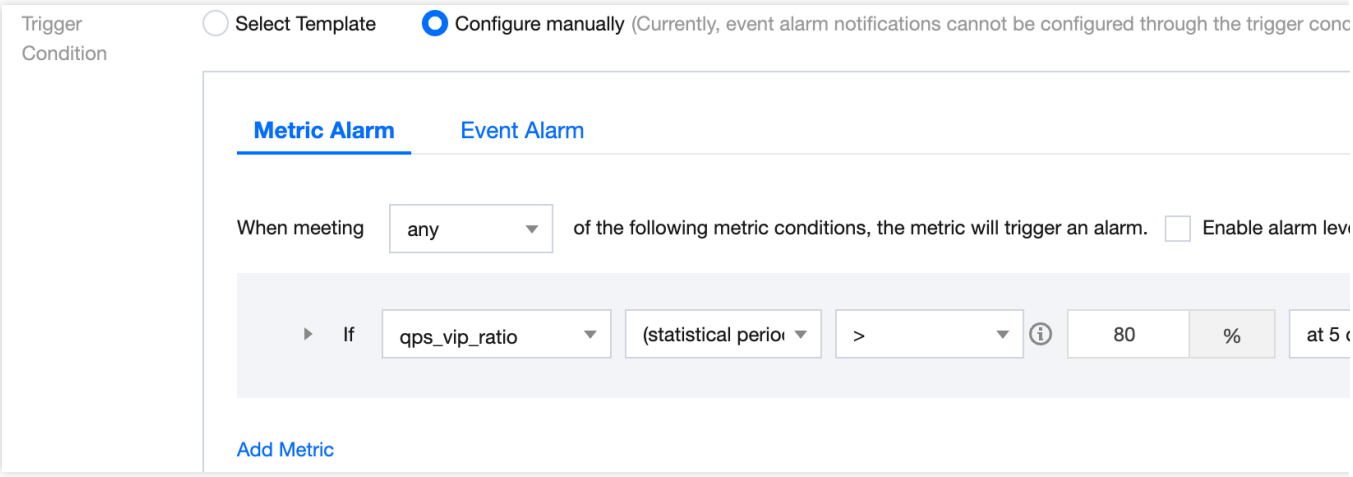
Options for configuration include IntrafficVipRatio, OuttrafficVipRatio, ConcurConnVipRatio, and NewConnVipRatio, as illustrated in the example below.



3.3 Configuring alarm notifications: Add a notification template, select an alarm recipient, notification cycle, and receiving channel. If a template has not been created, click **Create Template**. For detailed information, see Creating Notification Template.



4. Click **Done** to complete the configuration of monitoring alarms for IntrafficVipRatio, OuttrafficVipRatio, ConcurConnVipRatio, and NewConnVipRatio. For creating QPS usage monitoring alarms, refer to the previous step of creating a new alarm policy, modify the policy type to **CLB** > **Public CLB Instance** > **QPS Related Monitoring**, and configure the following trigger conditions.



Solutions

If you receive the preceding alarms, there is an increase in your business volume and the specification limit of the current standard type is to be reached and your business demands can no longer be met. Please proceed to Adjusting Specification of LCU-supported CLBs to ensure that your business operations remain unaffected.

Implementing HA Across Multiple AZs

Last updated：2024-01-04 14:39:00

Implementing HA Across Multiple AZs

CLB instances support disaster recovery across availability zones. For example, multiple clusters can be deployed in two availability zones in the same region: Guangzhou Zone 3 and Guangzhou Zone 4. This achieves disaster recovery across availability zones in the same region. With this feature, a CLB instance can forward frontend access traffic to other availability zones in the same region within 10s if the entire availability zone fails, restoring service capability.

FAQs and Use Cases

Question 1: If CLB instance `test1` is configured for Guangzhou Zone 3, what is the policy for inbound public network traffic of the client?

Guangzhou Zone 3 and Guangzhou Zone 4 have a pair of IP resource pools, which can be seen as IP resources with equivalent load balancing capability. Developers do not need to figure out between the master cluster and slave cluster. When they purchase a CLB instance and bind it to CVM, two sets of rules will be generated and written into the two clusters, achieving high availability.

Question 2: Suppose that CLB instance `test1` is configured for Guangzhou Zone 3 and bound to 100 real servers in each availability zone (Zone A and Zone B). During business operation, 1 million HTTP persistent connections (with TCP connections kept alive) are established in each zone. If the entire CLB cluster in Guangzhou Zone 3 fails and becomes unavailable, what will happen to the business?

When the CLB instance in Guangzhou Zone 3 fails, all current persistent connections will be closed, while non-persistent connections will not be affected. The disaster recovery architecture will automatically bind the 100 servers in each zone to the CLB instance in Guangzhou Zone 4 within 10s, immediately restoring business capability with no manual intervention required.

Question 3: Which type of CLB is compatible with multi-AZ disaster recovery? Does it cost extra fees?

The multi-AZ disaster recovery is in beta test, and no extra fees will be charged. Currently, only public network CLB supports multi-AZ disaster recovery. Different from the cross-AZ disaster recovery of public network CLB, private network CLB supports nearby access by default.

Load Balancing Algorithm Selection and Weight Configuration Examples

Last updated：2024-01-04 14:39:00

Comparative analysis of CLB algorithms

Weighted round-robin scheduling

The weighted round-robin scheduling algorithm is to schedule requests to different servers based on polling. It can solve problems with imbalanced performance of different servers. It uses weight to represent the processing performance of a server and schedules requests to different servers by weight in a polling manner. It schedules servers based on the number of new connections, where servers with a higher weight receive connections earlier and have a higher chance to be polled. Servers with the same weight will process the same number of connections.
Advantage: this algorithm features simplicity and high practicability. It does not need to record the status of all connections and is therefore a stateless scheduling algorithm.
Disadvantage: this algorithm is relatively simple, so it is unsuitable for situations where the service time of a request changes significantly, or each request needs to consume different amounts of time. In these cases, it will cause imbalanced load distribution among servers.
Applicable scenario: this algorithm is suitable for scenarios where each request consumes basically the same amount of time on the backend with the best loading performance. It is usually used in non-persistent connection services such as HTTP service.
Recommendation: If you know that each request consumes basically the same amount of time on the backend (for example, requests processed by a real server are of the same or similar types), we recommend you use weighted round-robin scheduling. If the time difference between each request is small, we recommend you use this algorithm because it has a low consumption, high efficiency, and no need for traversal.

Weighted least-connection scheduling

How it works

In actual situations, the time each client request spends on the server may vary greatly. If a simple round-robin or random load balancing algorithm is used as the working time gets longer, the number of connection processes on each server may vary greatly and load balancing may not be achieved. Contrary to round-robin scheduling, least connection scheduling is a dynamic scheduling algorithm that estimates the load on a server based on its number of active connections. The scheduler needs to record the number of connections currently established on each server. If a request is scheduled to a server, its number of connections increases by 1. If a connection stops or times out, its number of connections decreases by 1. The weighted least-connection scheduling algorithm is based on least-connection scheduling, and different weights are allocated to servers according to their processing performance.

Based on its weight, the server can then receive a corresponding number of requests. This algorithm is an improvement of least-connection scheduling.

1. Suppose the weight of a real server is Wi (i = 1... n) and its current number of connections is Ci (i = 1... n). The Ci/Wi value of each server is calculated in sequence. The real server with the smallest Ci/Wi value will be the next server to receive a new request.
2. For real servers with the same Ci/Wi value, they will be scheduled based on weighted round robin scheduling.

Advantages

This algorithm is suitable for requests requiring long-time processing, such as FTP.

Disadvantages

Due to API restrictions, least-connection and session persistence cannot be enabled at the same time.

Use cases

This algorithm is suitable for scenarios where the time used by each request on the backend varies greatly. It is usually used in persistence connection services.

Recommendations

If you need to process different requests and their service time on the backend varies greatly (such as 3 milliseconds and 3 seconds), we recommend you use weighted least-connection scheduling to achieve load balancing.

Source hashing scheduling (ip_hash)

How it works

Source hashing scheduling uses the source IP address of the request as the hash key and finds the corresponding server from the statically assigned hash table. The request will be sent to this server if it is available and not overloaded. Otherwise, null will be returned.

Advantages

ip_hash can achieve certain session persistence by remembering the source IP and mapping requests from a client to the same real server via the hash table. In scenarios where session persistence is not supported, ip_hash can be used for scheduling.

Recommendations

This algorithm calculates the hash value of the source address of a request and distributes the request to the corresponding real server based on its weight. This allows requests from the same client IP to be distributed to the same server. This algorithm is suitable for load balancing over TCP protocol that does not support cookie.

Load Balancing Algorithm Selection and Weight Configuration Examples

In the upcoming features of CLB, **Layer-7 forwarding will support the least-connection balancing method.** We provide examples for your reference on how to choose load balancing algorithm and configure weight, so you can ensure that real server clusters undertake business in different scenarios with stability.

Scenario 1

Suppose there are 3 real servers with the same configuration (CPU and memory) and you configure their weights all to 10. Suppose 100 TCP connections have been established between each real server and the client. If a new real server is added, we recommend you use the least connection scheduling algorithm, which can quickly increase the load of the 4th server and reduce the pressure on the other 3 servers.

Scenario 2

Suppose you use Tencent Cloud services for the first time. Your website has just been built and has low load. We recommend you purchase real servers of the same configuration because they are all access-layer servers. In this scenario, you can configure the weights of all real servers to 10 and use weighted round-robin scheduling algorithm to distribute traffic.

Scenario 3

Suppose you have 5 real servers to undertake simple access requests to static websites, and the ratio of their computing power (calculated by CPU and memory) is 9:3:3:3:1. In this scenario, you can configure the weights of servers to 90, 30, 30, 30, and 10 respectively. As access requests to static websites are mostly of non-persistence connection type, you can use weighted round-robin scheduling algorithm, so CLB can allocate requests based on the performance ratio of real servers.

Scenario 4

Suppose you have 10 real servers to undertake massive amounts of web access requests, and you do not want to purchase more servers. One of the servers often restarts due to overload. In this scenario, we recommend you configure the weights of existing servers based on their performance. Server with higher load should have a smaller weight. In addition, you can use least-connection scheduling algorithm to allocate requests to real servers with fewer active connections to avoid server overload.

Scenario 5

Suppose you have 3 real servers to process persistent connections, and the ratio of their computing power (calculated by CPU and memory) is 3:1:1. The server with the best performance processes more requests, but you do not want it to be overloaded. Instead, you want to allocate new requests to idle servers. In this scenario, you can use least connection scheduling algorithm and appropriately reduce the weights of busy servers, so CLB can allocate requests to real servers with fewer active connections, thereby achieving load balancing.

Scenario 6

Suppose you want subsequent requests from the client to be allocated to the same server. The weighted round-robin or weighted least-connection scheduling cannot ensure that requests from the same client are allocated to the same server. To meet the requirements of your specified application server and maintain the "stickiness" (or "continuity") of client sessions, we recommend you use ip_hash to distribute the traffic. This algorithm ensures that all requests from the same client will be distributed to the same real server, unless the number of servers changes or the server becomes unavailable.

Difference Between Resetting Weight to 0 and Unbinding Real Server

Resetting the weight to 0: TCP listeners keep forwarding existing connections, UDP listeners keep forwarding connections with the same quintuple, and HTTP/HTTPS listeners keep forwarding existing connections.New connections on TCP, UDP, HTTP/HTTPS listeners will no longer be forwarded to RS with a weight of zero.
Unbinding the real server: TCP/UDP listeners cease forwarding residual connections instantaneously, while HTTP/HTTPS listeners persist in forwarding residual connections. Upon completion of forwarding residual connections, the connection with RS is severed.

References

Managing Real Servers

Configuring WAF protection for CLB listening domain names

Last updated：2024-01-04 14:39:00

By binding domain names with CLB listeners, CLB Web Application Firewall (WAF) can detect and block the HTTP or HTTPS traffic passing through CLB listeners. This document describes how to use CLB WAF to apply Web security protection for the domain names added to CLB.

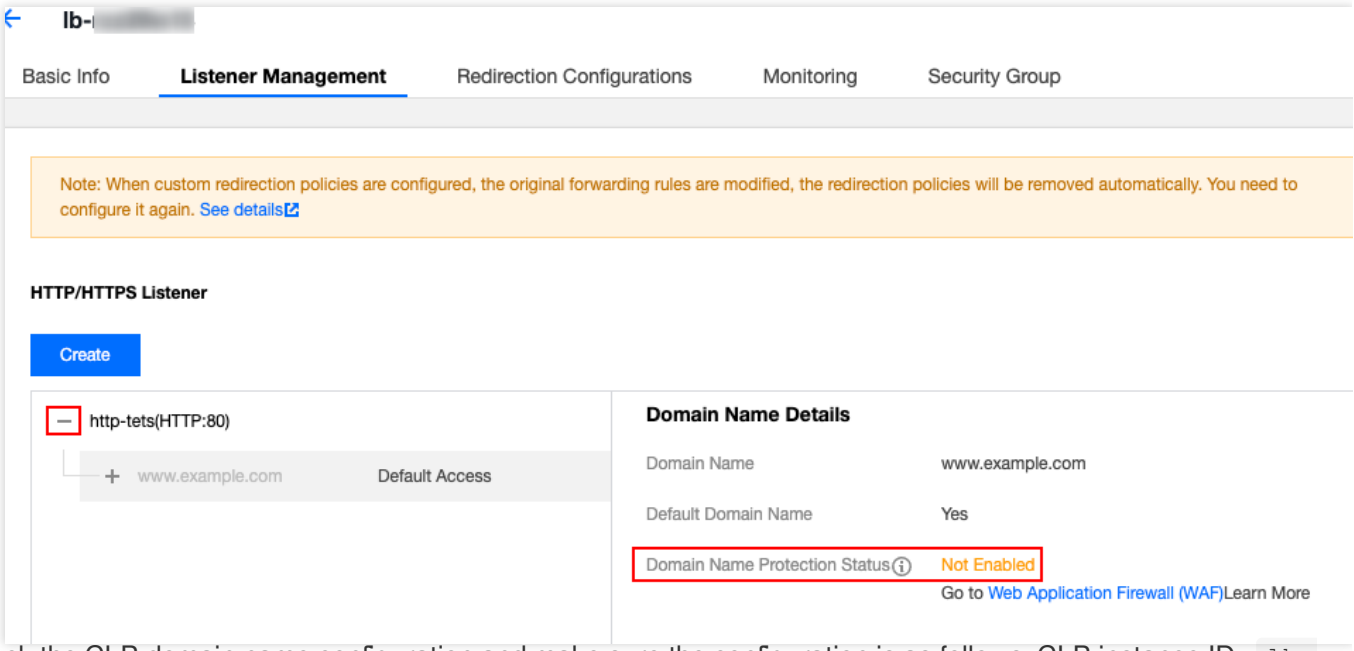
Prerequisites

You have created an HTTP or HTTPS listener, and the domain name can be accessed. For more information, see Getting Started with CLB.
You have purchased the CLB WAF service. For more information, see Purchase Guide.

Directions

Step 1: Confirm the CLB domain name configuration

This document takes the domain name `www.example.com` as an example.
1. Log in to the CLB console and click **Instance management** in the left sidebar.
2. On the **Instance management** page, select the instance region and then click **Configure listener** on the right of the target instance.
3. Select the **Listener management** tab, in the **HTTP/HTTPS listener** section, click the **+** icon on the left of the target listener to see the domain name details.

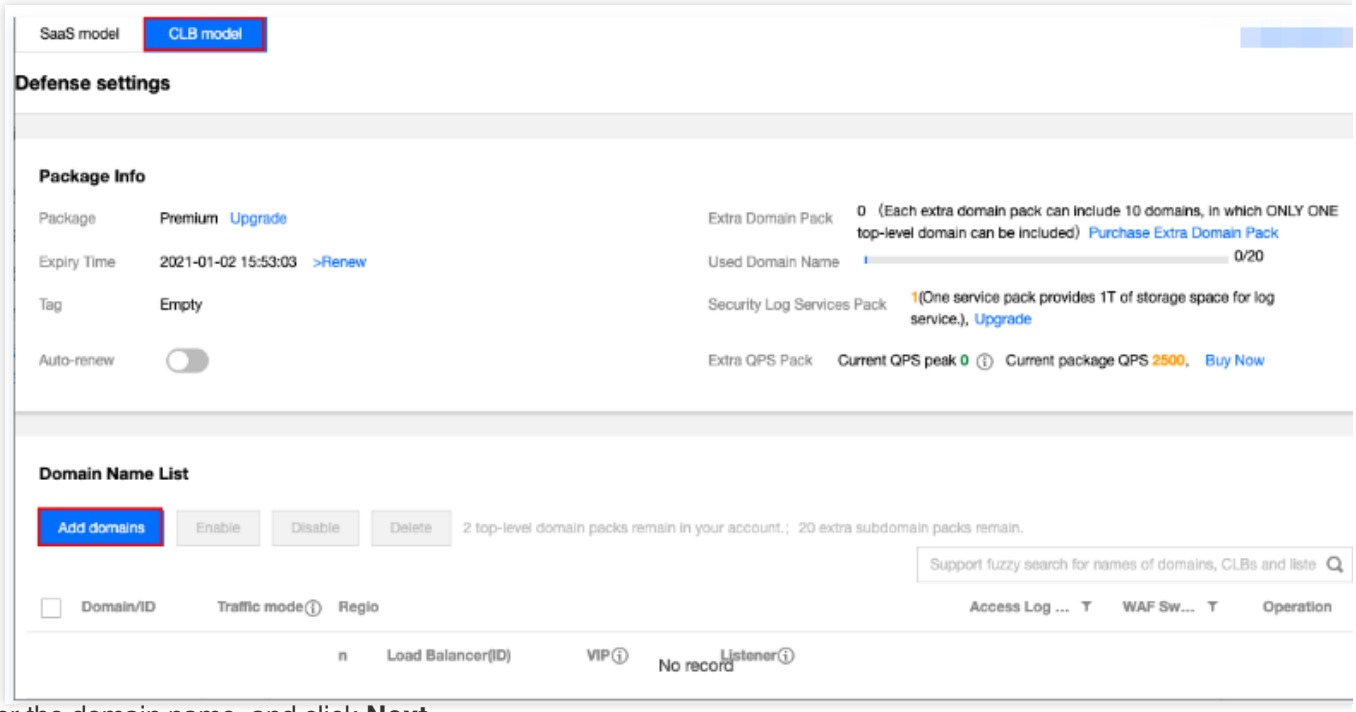


4. Check the CLB domain name configuration and make sure the configuration is as follows: CLB instance ID: `lb-f8lm****` ; listener name: `http-test` ; domain name: `www.example.com` ; domain name protection status: **Not Enabled** (the ID, name, and domain name are subject to actual cases).

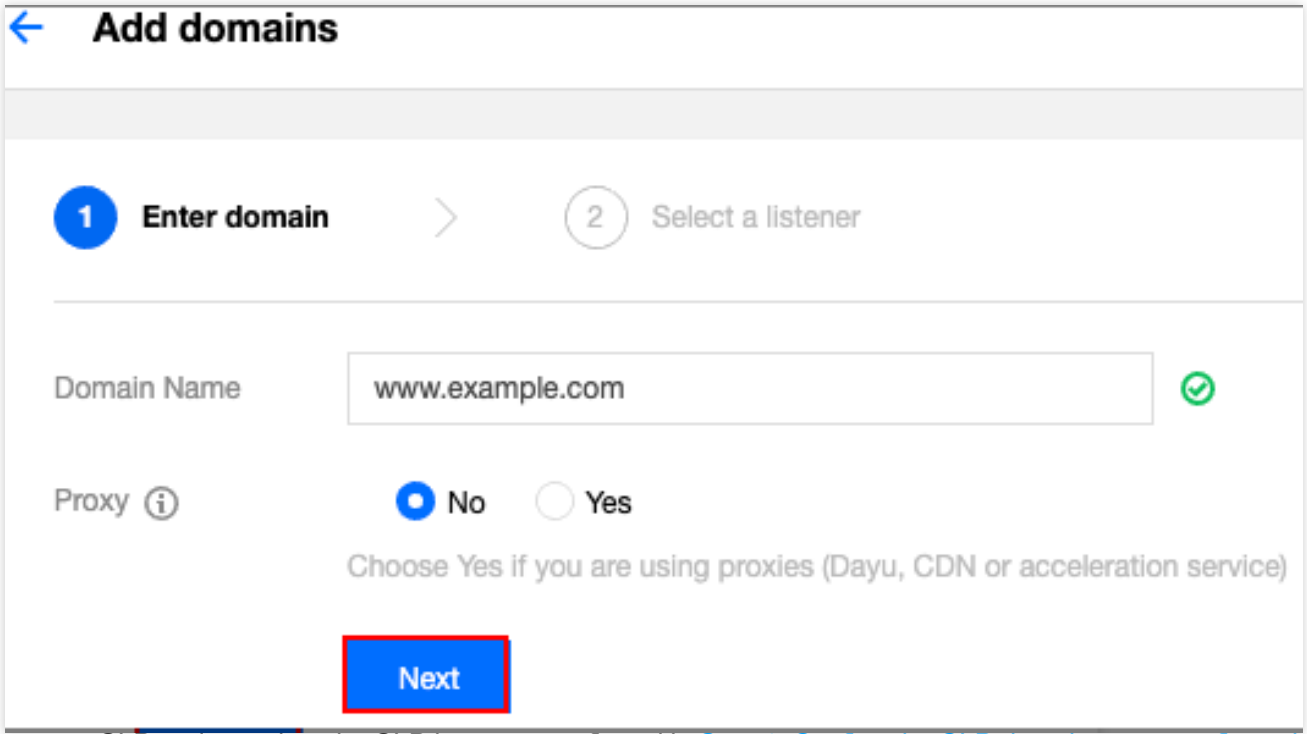
Step 2: Add a domain name in the WAF console and bind it to a CLB instance

To apply protection to a domain name with the CLB WAF service, you need to add a CLB-listening domain name in WAF and bind it with a CLB listener.

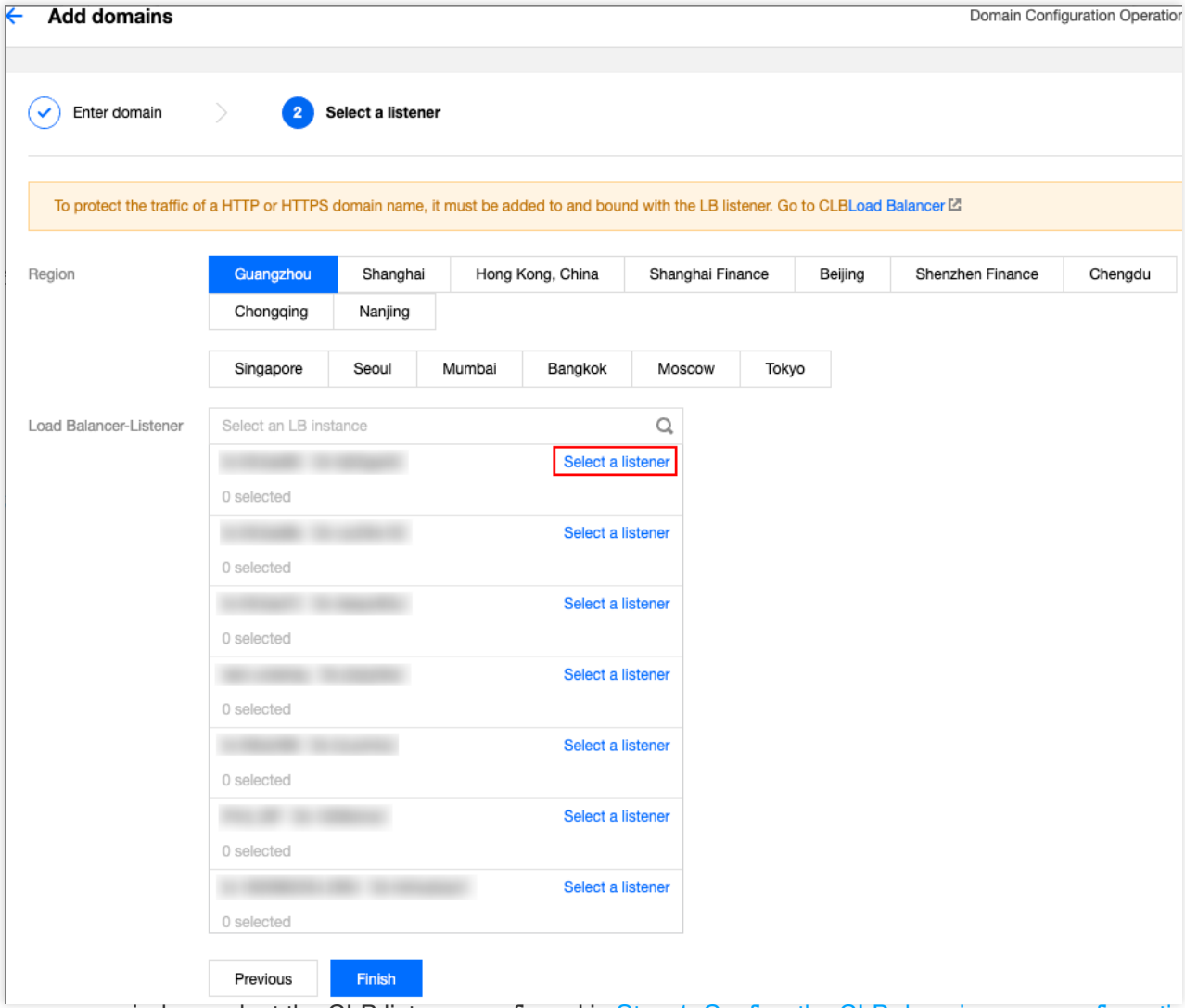
1. Log in to the WAF console, and choose **Web Application Firewall** > **Defense Settings** in the left sidebar.
2. Select the **CLB** tab.
3. Click **Add domain name**.



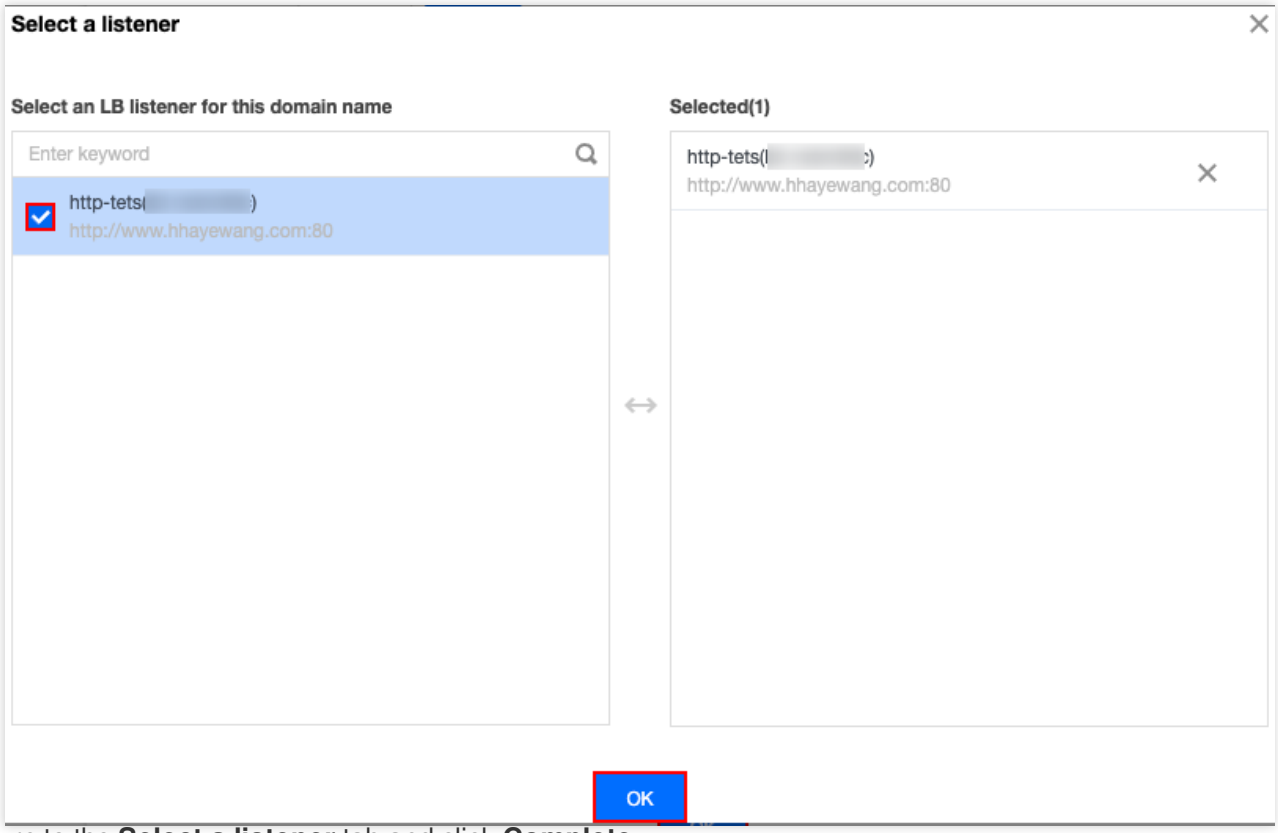
4. Enter the domain name, and click **Next**.



5. Select your CLB region, select the CLB instance confirmed in Step 1: Confirm the CLB domain name configuration, and click **Select a listener**.



6. In the pop-up window, select the CLB listener confirmed in Step 1: Confirm the CLB domain name configuration and click **OK**.



7. Return to the **Select a listener** tab and click **Complete**.

8. Return to the **Domain name list** page, check the domain name, region, ID of the bound CLB instance, bound listener, and other information.

Step 3: Verify the result

1. Follow the directions in Step 1: Confirm the CLB domain name configuration to check whether **Domain name protection** is **Enabled** and whether **Traffic mode** is **Traffic mirroring mode** on the **Listener management** tab. If so, domain name protection is enabled.

If you have not configured DNS resolution for your domain name, see Step 2. Perform Local Testing to verify if WAF protection takes effect.

If you have configured DNS resolution for your domain name, follow the directions below to verify if WAF protection takes effect.

2. Visit `http://www.example.com/?test=alert(123)` via a browser.

3. Log in to the WAF console, and choose **Attack Logs** in the left sidebar.

4. On the **Log Query** tab, select the protected domain name `www.example.com`, and then click **Search**. WAF protection takes effect on the domain name configured in CLB if there are **XSS attack** logs in the log list.