Tencent Kubernetes Engine

Fault Handling

Product Documentation

Contents

Fault Handling
Disk Full

Last updated：2022-04-20 19:12:22

This article describes the causes of the TKE cluster disk full issue. It also provides instructions on how to troubleshoot and solve this issue.

Possible Causes

kubelet uses gc and the eviction mechanism, along with parameters such as `--image-gc-high-threshold`, `--image-gc-low-threshold`, `--eviction-hard`, `--eviction-soft`, and `--eviction-minimum-reclaim`, to control and implement disk space reclamation. If not properly configured or if a non-Kubernetes process continues to write data to the disk, the disk will run out of space.

A full disk impacts the operation of Kubernetes, specifically two of its major components: kubelet and container runtime. Refer to the following instructions for troubleshooting:

1. Run `df` to check for kubelet and container runtime directories on the disk in question.
2. Use the results as a starting point for further troubleshooting.

- Container runtime directory is on a full disk
- kubelet directory is on a full disk

Troubleshooting

Container runtime directory is on a full disk

If the container runtime directory is on a full disk, it may lead to a non-responsive container runtime. For example, if you are using dockerd, docker commands will hang and kubelet logs will show “PLEG unhealthy”. CRI will then invoke timeout which leads to container creation or termination failures. In this case, the user will see that the Pod remains in the ContainerCreating or Terminating status.

Default Docker directories

- `/var/run/docker`: used to store container runtime statuses. You can use dockerd `-exec-root` to specify a different directory.
- `/var/lib/docker`: used to store persistent container data, such as container images, container writable layer data, container standard log output, and volumes created through Docker.

Pod launch process

The following is a sample Pod launch process:

```
Warning FailedCreatePodSandBox 53m kubelet, 172.22.0.44 Failed create pod sandbo
x: rpc error: code = DeadlineExceeded desc = context deadline exceeded
```

```
Warning FailedCreatePodSandBox 2m (x4307 over 16h) kubelet, 10.179.80.31 (combine
d from similar events): Failed create pod sandbox: rpc error: code = Unknown desc
= failed to create a sandbox for pod "apigateway-6dc48bf8b6-l8xrw": Error respons
e from daemon: mkdir /var/lib/docker/aufs/mnt/1f09d6c1c9f24e8daaea5bf33a4230de7db
c758e3b22785e8ee21e3e3d921214-init: no space left on device
```

```
Warning Failed 5m1s (x3397 over 17h) kubelet, ip-10-0-151-35.us-west-2.compute.in
ternal (combined from similar events): Error: container create failed: container_
linux.go:336: starting container process caused "process_linux.go:399: container
init caused \"rootfs_linux.go:58: mounting \\\"/sys\\\" to rootfs \\\"/var/lib/do
ckerd/storage/overlay/051e985771cc69f3f699895a1dada9ef6483e912b46a99e004af7bb4852
183eb/merged\\\" at \\\"/var/lib/dockerd/storage/overlay/051e985771cc69f3f699895a
1dada9ef6483e912b46a99e004af7bb4852183eb/merged/sys\\\" caused \\\"no space left
on device\\\"\""
```

Pod deletion process

The following is a sample Pod deletion process:

```
Normal Killing 39s (x735 over 15h) kubelet, 10.179.80.31 Killing container with i
d docker://apigateway:Need to kill Pod
```

kubelet directory is on a full disk

Default kubelet directories

`/var/lib/kubelet` : used to store plugin information, Pod statuses, and mounted volumes, such as `emptyDir` , `ConfigMap` , and `Secret` . You can use kubelet `--root-dir` to specify a different directory.

Pod creation process

If the kubetlet directory is on a full disk (usually the system disk), the Pod creation process stops at mkdir, which means Sandbox cannot be created. The following is a sample Pod creation process:

```
Warning UnexpectedAdmissionError 44m kubelet, 172.22.0.44 Update plugin resources
failed due to failed to write checkpoint file "kubelet_internal_checkpoint": writ
e /var/lib/kubelet/device-plugins/.728425055: no space left on device, which is u
nexpected.
```

Directions

If you use dockerd as the container runtime, a full disk causes dockerd to be unresponsive and hang when you try to stop it, which means you can't restart dockerd to release storage space. In this case, you need to perform a manual cleanup to free up enough space for dockerd to restart. The procedure is as follows:

1. Manually delete some of the log files or files on the writable layer, as shown below:

```
$ cd /var/lib/docker/containers
$ du -sh * # Find a directory that occupies a lot of space.
$ cd dda02c9a7491fa797ab730c1568ba06cba74cecd4e4a82e9d90d00fa11de743c
$ cat /dev/null > dda02c9a7491fa797ab730c1568ba06cba74cecd4e4a82e9d90d00fa11de7
43c-json.log.9 # Delete log files.
```

Note：

- We recommend that you use `cat /dev/null >` to delete files rather than `rm`. Files deleted using `rm` are not released by docker processes and therefore the space they occupy is not released.
- the larger the suffix number, the older the log file. We recommend that you delete older log files first.

2. Run the following command to mark the node as unschedulable and evict existing Pods to other nodes:

```
kubectl drain <node-name>
```

This ensures that the containers in the Pod of the original node are deleted, as well as their logs (standard output) and container data (unmounted volumes and writable layer).

3. Run the following command to restart dockerd:

```
systemctl restart dockerd
# or systemctl restart docker
```

4. After dockerd is restarted and the Pod is scheduled to another node, find the cause for the full disk, perform a data cleanup, and take prevention measures.
5. Run the following command to remove the unschedulable mark from the node:

```
kubectl uncordon <node-name>
```

How to Prevent Disks from Filling Up

Make sure kubelet gc and eviction parameters are properly configured. Once you have done that, even if the disk becomes full, the Pods on the problematic node can be evicted automatically to other nodes, which prevents them from remaining in the ContainerCreating or Terminating status.

High Workload

Last updated：2022-04-20 19:13:54

This article describes how to troubleshoot TKE cluster issues caused by high loads.

Error Description

High loads prevent node processes from getting the CPU time they need to function properly, which can lead to network timeout, health check failures, and service unavailability.

Troubleshooting

At times, a node's load increases even though cpu 'us' (user) is low and cpu 'id' (idle) is high. This is usually caused by file I/O bottlenecks, which results in excessive I/O wait. In turn, this leads to high loads and impacts the performance of other processes.
This article uses top, atop, and iotop to diagnose if the performance issue is caused by disk I/O bottlenecks.

Query average load and wait time

1. Log in to your node and use `top` to query the current load. The following results are displayed:

> Note：
> High `load average` means the node is handling a large amount of requests. You can use values in the `Cpu(s)`, `Mem`, `%CPU`, and `%MEM` columns to see which processes are using a large portion of the resources.

```
top - 19:42:06 up 23:59, 2 users, load average: 34.64, 35.80, 35.76
Tasks: 679 total, 1 running, 678 sleeping, 0 stopped, 0 zombie
Cpu(s): 15.6%us, 1.7%sy, 0.0%ni, 74.7%id, 7.9%wa, 0.0%hi, 0.1%si, 0.0%st
Mem: 32865032k total, 30989168k used, 1875864k free, 370748k buffers
Swap: 8388604k total, 5440k used, 8383164k free, 7982424k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
9783 mysql 20 0 17.3g 16g 8104 S 186.9 52.3 3752:33 mysqld
5700 nginx 20 0 1330m 66m 9496 S 8.9 0.2 0:20.82 php-fpm
6424 nginx 20 0 1330m 65m 8372 S 8.3 0.2 0:04.97 php-fpm
```

```
6573 nginx 20 0 1330m 64m 7368 S 8.3 0.2 0:01.49 php-fpm
5927 nginx 20 0 1320m 56m 9272 S 7.6 0.2 0:12.54 php-fpm
5956 nginx 20 0 1330m 65m 8500 S 7.6 0.2 0:12.70 php-fpm
6126 nginx 20 0 1321m 57m 8964 S 7.3 0.2 0:09.72 php-fpm
6127 nginx 20 0 1319m 54m 9520 S 6.6 0.2 0:08.73 php-fpm
6131 nginx 20 0 1320m 56m 9404 S 6.6 0.2 0:09.43 php-fpm
6174 nginx 20 0 1321m 56m 8444 S 6.3 0.2 0:08.92 php-fpm
5790 nginx 20 0 1319m 54m 9468 S 5.6 0.2 0:17.33 php-fpm
6575 nginx 20 0 1320m 55m 8212 S 5.6 0.2 0:02.11 php-fpm
6160 nginx 20 0 1310m 44m 8296 S 4.0 0.1 0:10.05 php-fpm
5597 nginx 20 0 1310m 46m 9556 S 3.6 0.1 0:21.03 php-fpm
5786 nginx 20 0 1310m 45m 8528 S 3.6 0.1 0:15.53 php-fpm
5797 nginx 20 0 1310m 46m 9444 S 3.6 0.1 0:14.02 php-fpm
6158 nginx 20 0 1310m 45m 8324 S 3.6 0.1 0:10.20 php-fpm
5698 nginx 20 0 1310m 46m 9184 S 3.3 0.1 0:20.62 php-fpm
5779 nginx 20 0 1309m 44m 8336 S 3.3 0.1 0:15.34 php-fpm
6540 nginx 20 0 1306m 40m 7884 S 3.3 0.1 0:02.46 php-fpm
5553 nginx 20 0 1300m 36m 9568 S 3.0 0.1 0:21.58 php-fpm
5722 nginx 20 0 1310m 45m 8552 S 3.0 0.1 0:17.25 php-fpm
5920 nginx 20 0 1302m 36m 8208 S 3.0 0.1 0:14.23 php-fpm
6432 nginx 20 0 1310m 45m 8420 S 3.0 0.1 0:05.86 php-fpm
5285 nginx 20 0 1302m 38m 9696 S 2.7 0.1 0:23.41 php-fpm
```

2. Among the results is the CPU `wa` value. `wa` (wait) is the percent of CPU resources used by IO WAIT. By default, the result shows the average value of all cores. Press **1** to view the `wa` value of each core, as shown below:

> Note：
> `wa` is usually 0%. If it constantly floats above 1%, this indicates a storage bottleneck has been reached and storage cannot keep up with CPU processing speed.

```
top - 19:42:08 up 23:59, 2 users, load average: 34.64, 35.80, 35.76
Tasks: 679 total, 1 running, 678 sleeping, 0 stopped, 0 zombie
Cpu0 : 29.5%us, 3.7%sy, 0.0%ni, 48.7%id, 17.9%wa, 0.0%hi, 0.1%si, 0.0%st
Cpu1 : 29.3%us, 3.7%sy, 0.0%ni, 48.9%id, 17.9%wa, 0.0%hi, 0.1%si, 0.0%st
Cpu2 : 26.1%us, 3.1%sy, 0.0%ni, 64.4%id, 6.0%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu3 : 25.9%us, 3.1%sy, 0.0%ni, 65.5%id, 5.4%wa, 0.0%hi, 0.1%si, 0.0%st
Cpu4 : 24.9%us, 3.0%sy, 0.0%ni, 66.8%id, 5.0%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu5 : 24.9%us, 2.9%sy, 0.0%ni, 67.0%id, 4.8%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu6 : 24.2%us, 2.7%sy, 0.0%ni, 68.3%id, 4.5%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu7 : 24.3%us, 2.6%sy, 0.0%ni, 68.5%id, 4.2%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu8 : 23.8%us, 2.6%sy, 0.0%ni, 69.2%id, 4.1%wa, 0.0%hi, 0.3%si, 0.0%st
```

```
Cpu9 : 23.9%us, 2.5%sy, 0.0%ni, 69.3%id, 4.0%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu10 : 23.3%us, 2.4%sy, 0.0%ni, 68.7%id, 5.6%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu11 : 23.3%us, 2.4%sy, 0.0%ni, 69.2%id, 5.1%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu12 : 21.8%us, 2.4%sy, 0.0%ni, 60.2%id, 15.5%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu13 : 21.9%us, 2.4%sy, 0.0%ni, 60.6%id, 15.2%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu14 : 21.4%us, 2.3%sy, 0.0%ni, 72.6%id, 3.7%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu15 : 21.5%us, 2.2%sy, 0.0%ni, 73.2%id, 3.1%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu16 : 21.2%us, 2.2%sy, 0.0%ni, 73.6%id, 3.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu17 : 21.2%us, 2.1%sy, 0.0%ni, 73.8%id, 2.8%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu18 : 20.9%us, 2.1%sy, 0.0%ni, 74.1%id, 2.9%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu19 : 21.0%us, 2.1%sy, 0.0%ni, 74.4%id, 2.5%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu20 : 20.7%us, 2.0%sy, 0.0%ni, 73.8%id, 3.4%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu21 : 20.8%us, 2.0%sy, 0.0%ni, 73.9%id, 3.2%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu22 : 20.8%us, 2.0%sy, 0.0%ni, 74.4%id, 2.8%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu23 : 20.8%us, 1.9%sy, 0.0%ni, 74.4%id, 2.8%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 32865032k total, 30209248k used, 2655784k free, 370748k buffers
Swap: 8388604k total, 5440k used, 8383164k free, 7986552k cached
```

Monitoring Disk I/O Statistics

1. Use `atop` to query disk I/O. In the following example, disk `sda` shows `busy 100%`, meaning it has reached the bottleneck.

```
ATOP - lemp 2017/01/23 19:42:32 --------- 10s elapsed
PRC | sys 3.18s | user 33.24s | #proc 679 | #tslpu 28 | #zombie 0 | #exit 0 |
CPU | sys 29% | user 330% | irq 1% | idle 1857% | wait 182% | curscal 69% |
CPL | avg1 33.00 | avg5 35.29 | avg15 35.59 | csw 62610 | intr 76926 | numcpu 2
4 |
MEM | tot 31.3G | free 2.1G | cache 7.6G | dirty 41.0M | buff 362.1M | slab 1.2
G |
SWP | tot 8.0G | free 8.0G | | | vmcom 23.9G | vmlim 23.7G |
DSK | sda | busy 100% | read 4 | write 1789 | MBw/s 2.84 | avio 5.58 ms |
NET | transport | tcpi 10357 | tcpo 9065 | udpi 0 | udpo 0 | tcpao 174 |
NET | network | ipi 10360 | ipo 9065 | ipfrw 0 | deliv 10359 | icmpo 0 |
NET | eth0 4% | pcki 6649 | pcko 6136 | si 1478 Kbps | so 4115 Kbps | erro 0 |
NET | lo ---- | pcki 4082 | pcko 4082 | si 8967 Kbps | so 8967 Kbps | erro 0 |
PID TID THR SYSCPU USRCPU VGROW RGROW RDDSK WRDSK ST EXC S CPUNR CPU CMD 1/12
9783 - 156 0.21s 19.44s 0K -788K 4K 1344K -- - S 4 197% mysqld
5596 - 1 0.10s 0.62s 47204K 47004K 0K 220K -- - S 18 7% php-fpm
6429 - 1 0.06s 0.34s 19840K 19968K 0K 0K -- - S 21 4% php-fpm
6210 - 1 0.03s 0.30s -5216K -5204K 0K 0K -- - S 19 3% php-fpm
5757 - 1 0.05s 0.27s 26072K 26012K 0K 4K -- - S 13 3% php-fpm
6433 - 1 0.04s 0.28s -2816K -2816K 0K 0K -- - S 11 3% php-fpm
5846 - 1 0.06s 0.22s -2560K -2660K 0K 0K -- - S 7 3% php-fpm
5791 - 1 0.05s 0.21s 5764K 5692K 0K 0K -- - S 22 3% php-fpm
```

```
5860 - 1 0.04s 0.21s 48088K 47724K 0K 0K -- - S 1 3% php-fpm
6231 - 1 0.04s 0.20s -256K -4K 0K 0K -- - S 1 2% php-fpm
6154 - 1 0.03s 0.21s -3004K -3184K 0K 0K -- - S 21 2% php-fpm
6573 - 1 0.04s 0.20s -512K -168K 0K 0K -- - S 4 2% php-fpm
6435 - 1 0.04s 0.19s -3216K -2980K 0K 0K -- - S 15 2% php-fpm
5954 - 1 0.03s 0.20s 0K 164K 0K 4K -- - S 0 2% php-fpm
6133 - 1 0.03s 0.19s 41056K 40432K 0K 0K -- - S 18 2% php-fpm
6132 - 1 0.02s 0.20s 37836K 37440K 0K 0K -- - S 11 2% php-fpm
6242 - 1 0.03s 0.19s -12.2M -12.3M 0K 4K -- - S 12 2% php-fpm
6285 - 1 0.02s 0.19s 39516K 39420K 0K 0K -- - S 3 2% php-fpm
6455 - 1 0.05s 0.16s 29008K 28560K 0K 0K -- - S 14 2% php-fpm
```

2. Use one of the following methods to view process disk I/O usage:

   - Press **d** to view process disk I/O usage, as shown below:

```
ATOP - lemp 2017/01/23 19:42:46 --------- 2s elapsed
PRC | sys 0.24s | user 1.99s | #proc 679 | #tslpu 54 | #zombie 0 | #exit 0 |
CPU | sys 11% | user 101% | irq 1% | idle 2089% | wait 208% | curscal 63% |
CPL | avg1 38.49 | avg5 36.48 | avg15 35.98 | csw 4654 | intr 6876 | numcpu 2
4 |
MEM | tot 31.3G | free 2.2G | cache 7.6G | dirty 48.7M | buff 362.1M | slab
1.2G |
SWP | tot 8.0G | free 8.0G | | | vmcom 23.9G | vmlim 23.7G |
DSK | sda | busy 100% | read 2 | write 362 | MBw/s 2.28 | avio 5.49 ms |
NET | transport | tcpi 1031 | tcpo 968 | udpi 0 | udpo 0 | tcpao 45 |
NET | network | ipi 1031 | ipo 968 | ipfrw 0 | deliv 1031 | icmpo 0 |
NET | eth0 1% | pcki 558 | pcko 508 | si 762 Kbps | so 1077 Kbps | erro 0 |
NET | lo ---- | pcki 406 | pcko 406 | si 2273 Kbps | so 2273 Kbps | erro 0 |
PID TID RDDSK WRDSK WCANCL DSK CMD 1/5
9783 - 0K 468K 16K 40% mysqld
1930 - 0K 212K 0K 18% flush-8:0
5896 - 0K 152K 0K 13% nginx
880 - 0K 148K 0K 13% jbd2/sda5-8
5909 - 0K 60K 0K 5% nginx
5906 - 0K 36K 0K 3% nginx
5907 - 16K 8K 0K 2% nginx
5903 - 20K 0K 0K 2% nginx
5901 - 0K 12K 0K 1% nginx
5908 - 0K 8K 0K 1% nginx
5894 - 0K 8K 0K 1% nginx
5911 - 0K 8K 0K 1% nginx
5900 - 0K 4K 4K 0% nginx
5551 - 0K 4K 0K 0% php-fpm
```

```
5913 - 0K 4K 0K 0% nginx
5895 - 0K 4K 0K 0% nginx
6133 - 0K 0K 0K 0% php-fpm
5780 - 0K 0K 0K 0% php-fpm
6675 - 0K 0K 0K 0% atop
```

- You can also use `iotop -oPa` to view process disk I/O usage, as shown below:

```
Total DISK READ: 15.02 K/s | Total DISK WRITE: 3.82 M/s
PID PRIO USER DISK READ DISK WRITE SWAPIN IO> COMMAND
1930 be/4 root 0.00 B 1956.00 K 0.00 % 83.34 % [flush-8:0]
5914 be/4 nginx 0.00 B 0.00 B 0.00 % 36.56 % nginx: cache manager process
880 be/3 root 0.00 B 21.27 M 0.00 % 35.03 % [jbd2/sda5-8]
5913 be/2 nginx 36.00 K 1000.00 K 0.00 % 8.94 % nginx: worker process
5910 be/2 nginx 0.00 B 1048.00 K 0.00 % 8.43 % nginx: worker process
5896 be/2 nginx 56.00 K 452.00 K 0.00 % 6.91 % nginx: worker process
5909 be/2 nginx 20.00 K 1144.00 K 0.00 % 6.24 % nginx: worker process
5890 be/2 nginx 48.00 K 692.00 K 0.00 % 6.07 % nginx: worker process
5892 be/2 nginx 84.00 K 736.00 K 0.00 % 5.71 % nginx: worker process
5901 be/2 nginx 20.00 K 504.00 K 0.00 % 5.46 % nginx: worker process
5899 be/2 nginx 0.00 B 596.00 K 0.00 % 5.14 % nginx: worker process
5897 be/2 nginx 28.00 K 1388.00 K 0.00 % 4.90 % nginx: worker process
5908 be/2 nginx 48.00 K 700.00 K 0.00 % 4.43 % nginx: worker process
5905 be/2 nginx 32.00 K 1140.00 K 0.00 % 4.36 % nginx: worker process
5900 be/2 nginx 0.00 B 1208.00 K 0.00 % 4.31 % nginx: worker process
5904 be/2 nginx 36.00 K 1244.00 K 0.00 % 2.80 % nginx: worker process
5895 be/2 nginx 16.00 K 780.00 K 0.00 % 2.50 % nginx: worker process
5907 be/2 nginx 0.00 B 1548.00 K 0.00 % 2.43 % nginx: worker process
5903 be/2 nginx 36.00 K 1032.00 K 0.00 % 2.34 % nginx: worker process
6130 be/4 nginx 0.00 B 72.00 K 0.00 % 2.18 % php-fpm: pool www
5906 be/2 nginx 12.00 K 844.00 K 0.00 % 2.10 % nginx: worker process
5889 be/2 nginx 40.00 K 1164.00 K 0.00 % 2.00 % nginx: worker process
5894 be/2 nginx 44.00 K 760.00 K 0.00 % 1.61 % nginx: worker process
5902 be/2 nginx 52.00 K 992.00 K 0.00 % 1.55 % nginx: worker process
5893 be/2 nginx 64.00 K 972.00 K 0.00 % 1.22 % nginx: worker process
5814 be/4 nginx 36.00 K 44.00 K 0.00 % 1.06 % php-fpm: pool www
6159 be/4 nginx 4.00 K 4.00 K 0.00 % 1.00 % php-fpm: pool www
5693 be/4 nginx 0.00 B 4.00 K 0.00 % 0.86 % php-fpm: pool www
5912 be/2 nginx 68.00 K 300.00 K 0.00 % 0.72 % nginx: worker process
5911 be/2 nginx 20.00 K 788.00 K 0.00 % 0.72 % nginx: worker process
```

Use `man iotop` to view the descriptions of the following parameters:

```
-o, --only
Only show processes or threads actually doing I/O, instead of showing all process
es or threads. This can be dynamically toggled by pressing o.
-P, --processes
Only show processes. Normally iotop shows all threads.
-a, --accumulated
Show accumulated I/O instead of bandwidth. In this mode, iotop shows the amount o
f I/O processes have done since iotop started.
```

Other Reasons

Deploying non-Kubernetes services, such as databases, on the node may also cause high loads.

Memory Fragmentation

Last updated：2022-04-18 14:35:05

This article describes how to identify if a TKE cluster issue is caused by memory fragmentation and how to troubleshoot it.

Problem Analysis

If memory page allocation fails, the memory kernel outputs the following error message:

```
mysqld: page allocation failure. order:4, mode:0x10c0d0
```

- `mysqld` : application requesting memory.
- `order` : number of requested sequential memory pages (2^order). This example has an order of 4, which means 2^4 = 16 sequential pages.
- `mode` : memory allocation mode marker. This is defined in the kernel source code file `include/linux/gfp.h` and usually the result of the AND operation on multiple markers. Different kernels have different mode markers. For example, `GFP_KERNEL` in the new kernel is the result of `__GFP_RECLAIM | __GFP_IO | __GFP_FS` , and `__GFP_RECLAIM` is the result of `___GFP_DIRECT_RECLAIM | ___GFP_KSWAPD_RECLAIM` .

> Note：
>
> - When the value of order is 0, the system has no available memory.
> - When the value of order is large, the memory is fragmented, and no sequential large memory page can be allocated.

Error Description

Container fails to launch

Kubernetes creates netns for each Pod to isolate the network namespace. When the kernel initializes netns, it creates a cache for the nf_conntrack table, which needs large memory pages. If system memory is already fragmented, kernel will output the following error message due to the failure to allocate large memory pages (`v2.6.33 - v4.6`):

```
runc:[1:CHILD]: page allocation failure: order:6, mode:0x10c0d0
```

The pod remains in the ContainerCreating status and dockerd fails to launch containers. The following are the related log entries:

```
Jan 23 14:15:31 dc05 dockerd: time="2019-01-23T14:15:31.288446233+08:00" level=er
ror msg="containerd: start container" error="oci runtime error: container_linux.g
o:247: starting container process caused \"process_linux.go:245: running exec set
ns process for init caused \\\"exit status 6\\\"\"\n" id=5b9be8c5bb121264899fac8d
9d36b02150269d41ce96ba6ad36d70b8640cb01c
Jan 23 14:15:31 dc05 dockerd: time="2019-01-23T14:15:31.317965799+08:00" level=er
ror msg="Create container failed with error: invalid header field value \"oci run
time error: container_linux.go:247: starting container process caused \\\"process
_linux.go:245: running exec setns process for init caused \\\\\\\"exit status 6
\\\\\\\"\\\"\\n\""
```

kubelet log entries are as follows:

```
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352386 26037 remote_runtime.go:91] R
unPodSandbox from runtime service failed: rpc error: code = 2 desc = failed to st
art sandbox container for pod "matchdataserver-1255064836-t4b2w": Error response
from daemon: {"message":"invalid header field value \"oci runtime error: containe
r_linux.go:247: starting container process caused \\\"process_linux.go:245: runni
ng exec setns process for init caused \\\\\\\"exit status 6\\\\\\\"\\\"\\n\""}
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352496 26037 kuberuntime_sandbox.go:
54] CreatePodSandbox for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed
6-11e9-8661-0a587f8021ea)" failed: rpc error: code = 2 desc = failed to start san
dbox container for pod "matchdataserver-1255064836-t4b2w": Error response from da
emon: {"message":"invalid header field value \"oci runtime error: container_linu
x.go:247: starting container process caused \\\"process_linux.go:245: running exe
c setns process for init caused \\\\\\\"exit status 6\\\\\\\"\\\"\\n\""}
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352518 26037 kuberuntime_manager.go:
618] createPodSandbox for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1e
d6-11e9-8661-0a587f8021ea)" failed: rpc error: code = 2 desc = failed to start sa
ndbox container for pod "matchdataserver-1255064836-t4b2w": Error response from d
aemon: {"message":"invalid header field value \"oci runtime error: container_linu
x.go:247: starting container process caused \\\"process_linux.go:245: running exe
c setns process for init caused \\\\\\\"exit status 6\\\\\\\"\\\"\\n\""}
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352580 26037 pod_workers.go:182] Err
or syncing pod 485fd485-1ed6-11e9-8661-0a587f8021ea ("matchdataserver-1255064836-
t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)"), skipping: failed to "CreateP
odSandbox" for "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a
587f8021ea)" with CreatePodSandboxError: "CreatePodSandbox for pod \"matchdataser
ver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)\" failed: rpc er
ror: code = 2 desc = failed to start sandbox container for pod \"matchdataserver-
1255064836-t4b2w\": Error response from daemon: {\"message\":\"invalid header fie
ld value \\\"oci runtime error: container_linux.go:247: starting container proces
```

```
s caused \\\\\\\"process_linux.go:245: running exec setns process for init caused
\\\\\\\\\\\\\\\\"exit status 6\\\\\\\\\\\\\\\\"\\\\\\\"\\\\n\\\"\"}"
Jan 23 14:15:31 dc05 kubelet: I0123 14:15:31.372181 26037 kubelet.go:1916] SyncLo
op (PLEG): "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f
8021ea)", event: &pleg.PodLifecycleEvent{ID:"485fd485-1ed6-11e9-8661-0a587f8021e
a", Type:"ContainerDied", Data:"5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad
36d70b8640cb01c"}
Jan 23 14:15:31 dc05 kubelet: W0123 14:15:31.372225 26037 pod_container_deletor.g
o:77] Container "5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01
c" not found in pod's containers
Jan 23 14:15:31 dc05 kubelet: I0123 14:15:31.678211 26037 kuberuntime_manager.go:
383] No ready sandbox for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1e
d6-11e9-8661-0a587f8021ea)" can be found. Need to start a new one
```

Use `cat /proc/buddyinfo` to view slab. If there is no large memory available, you will see a lot of 0s, as shown below:

```
$ cat /proc/buddyinfo
Node 0, zone DMA 1 0 1 0 2 1 1 0 1 1 3
Node 0, zone DMA32 2725 624 489 178 0 0 0 0 0 0 0
Node 0, zone Normal 1163 1101 932 222 0 0 0 0 0 0 0
```

System OOM

Memory fragmentation leads to a lack of large memory pages. This causes application memory allocation failures even though there is plenty of system memory available. The system will assume it is out of memory and try to terminate processes in order to release memory, which leads to system OOM errors.

Directions

1. Periodically drop the cache or do so when there is a shortage of large memory pages.

```
echo 3 > /proc/sys/vm/drop_caches
```

2. Run the following command to compact the memory:

> Note：
> This operation is resource intensive and may cause business interruptions.

```
echo 1 > /proc/sys/vm/compact_memory
```

Cluster DNS Troubleshooting

Last updated：2023-10-08 17:53:41

Troubleshooting Approaches

1. Make sure that the cluster DNS runs normally

Container DNS passes through cluster DNS (usually CoreDNS). First, make sure that the cluster DNS runs normally. You can see the cluster IP of the DNS in the `--cluster-dns` startup parameter of kubelet:

```
$ ps -ef | grep kubelet
... /usr/bin/kubelet --cluster-dns=172.16.14.217 ...
```

Find the DNS Service:

```
$ kubectl get svc -n kube-system | grep 172.16.14.217
kube-dns ClusterIP 172.16.14.217 <none> 53/TCP,53/UDP 47d
```

Check for the endpoint:

```
$ kubectl -n kube-system describe svc kube-dns | grep -i endpoints
Endpoints: 172.16.0.156:53,172.16.0.167:53
Endpoints: 172.16.0.156:53,172.16.0.167:53
```

Check whether the Pod of the endpoint is normal:

```
$ kubectl -n kube-system get pod -o wide | grep 172.16.0.156
kube-dns-898dbbfc6-hvwlr 3/3 Running 0 8d 172.16.0.156 10.0.0.3
```

2. Make sure that the Pod can communicate with the cluster DNS

Check whether the Pod can connect to the cluster DNS. You can run the `telnet` command in the Pod to view port 53 of the DNS:

```
# Cluster IP for connecting to the DNS Service
$ telnet 172.16.14.217 53
```

Note：

If there are no testing tools such as telnet in the container, you can use nsenter to enter netns for packet capturing and use telnet on the host for testing.

If the network is found to be disconnected, check the following network settings:

- Check the security group settings of the node and the container IP range of the cluster that needs to be opened.
- Check for firewall rules and check the iptables.
- Check whether kube-proxy runs normally. The cluster DNS IP is the cluster IP, which is forwarded through the iptables or IPVS rules generated by kube-proxy.

3. Capture packets

If the cluster DNS runs normally and the Pod can communicate with the cluster DNS, capture packets for further checks. If the problem can be easily reproduced, you can use nsenter to enter netns to capture container packets:

```
tcpdump -i any port 53 -w dns.pcap
# tcpdump -i any port 53 -nn -tttt
```

If the cause still cannot be identified, you can capture packets at multiple points along the request linkage for analysis, such as Pod container, host cbr0 bridge, primary ENI of the host (eth0), primary ENI of the host of the CoreDNS Pod, cbr0, and container. Wait for the problem to recur and locate the point where the packet is lost.

Issue and Cause

Latency of five seconds

If it often takes five seconds to return a DNS query result, packets are usually lost due to kernel conntrack conflicts. The root cause is the bug in the conntrack module, where some packets are discarded due to resource competition when netfilter is used for NAT.

- It may possibly occur when multiple threads or processes send the same quintuple UDP packet through the same socket concurrently.
- Both glibc and musl (Alpine Linux's libc) use "parallel query", i.e., multiple query requests are sent concurrently, which tends to cause conflicts and request discarding.
- As IPVS also uses conntrack, this problem cannot be avoided in IPVS mode of kube-proxy.

Workaround

Use local DNS. DNS requests of the container are sent to the local DNS cache service (dnsmasq, nscd, etc.), without DNAT or conntrack conflicts. In addition, the DNS service will not be a performance bottleneck.
You can use local DNS in two ways:

- Each container comes with a DNS cache service.
- Each node runs a DNS cache service, and all containers use the DNS cache of the node as their nameserver.

Timeout of the resolution of an external domain name

Possible reasons:

- The upstream DNS fails.
- The ACL or firewall of the upstream DNS blocks the packet.

Timeout of all resolutions

If a cluster Pod fails to resolve both Services and external domain names, there is generally a problem with the communication between the Pod and the cluster DNS.
Possible reasons:

- The node firewall doesn't open the cluster IP range; therefore, the Pod cannot communicate with that of the cluster DNS when they are on different nodes, and DNS requests cannot be received.
- kube-proxy is abnormal.

Cluster kube-proxy Troubleshooting

Last updated：2022-11-02 14:35:48

TKE cluster access may fail in some cases. If you have confirmed that the backend Pod is normal, the cause may be that the kube-proxy add-on version is earlier than required, preventing iptables or IPVS forwarding rules on the node from being delivered successfully. This document describes some problems due to earlier kube-proxy versions and offers fixes. If you still have problems, contact us for assistance.

kube-proxy was not correctly adapted to the iptables backend of the node

Sample error message

```
Failed to execute iptables-restore: exit status 2 (iptables-restore v1.8.4 (legac
y): Couldn't load target 'KUBE-MARK-DROP':No such file or directory
```

Cause

1. When `iptables-restore` is executed in kube-proxy, the dependent `KUBE-MARK-DROP` chain doesn't exist, leading to the rule sync failure and exit. The `KUBE-MARK-DROP` chain is maintained by kubelet.
2. On some later OS versions, the iptables backend is nft; while on earlier kube-proxy versions, the iptables backend is legacy. When kube-proxy on an earlier version runs on OS on a later version, the iptables backend cannot be matched, and the `KUBE-MARK-DROP` chain cannot be read. Later OS versions include:
   - TLinux 2.6 (TK4)
   - TLinux 3.1
   - TLinux 3.2
   - CentOS 8
   - Ubuntu 20

Fix guide

Upgrade kube-proxy. Below is the sample logic:

TKE Cluster Version	Fix Policy
> 1.18	No fixes are required, as the problem doesn't exist.
1.18	Upgrade kube-proxy to v1.18.4-tke.26 or later.

TKE Cluster Version	Fix Policy
1.16	Upgrade kube-proxy to v1.16.3-tke.28 or later.
1.14	Upgrade kube-proxy to v1.14.3-tke.27 or later.
1.12	Upgrade kube-proxy to v1.12.4-tke.31 or later.
1.10	Upgrade kube-proxy to v1.10.5-tke.20 or later.

Note：
For more information on the latest TKE versions, see TKE Kubernetes Revision Version History.

iptables lock of kube-proxy

Concurrent write failure due to no iptables lock mounted to another add-on

Sample error message

```
Failed to execute iptables-restore: exit status 1 (iptables-restore: line xxx fai
led)
```

Cause

1. When writing iptables rules to the kernel, iptables commands (such as `iptables-restore`) will use a file lock for sync to avoid concurrent writes of multiple instances. On Linux, the file is generally `/run/xtables.lock` .
2. For a Pod that needs to call iptables commands, such as kube-proxy, kube-router, or HostNetwork Pod on the client, if the file is not mounted, the above problem of concurrent writes may occur.

Fix guide

For a Pod that needs to call iptables commands, you need to mount the host `/run/xtables.lock` file to the Pod as follows:

```
volumeMounts:
- mountPath: /run/xtables.lock
name: xtables-lock
readOnly: false
volumes:
```

```
- hOStPath:
path: /run/xtables.lock
type: FileOrCreate
name: xtables-lock
```

Writes are blocked due to an earlier iptables-restore version

Sample error message

```
Failed to execute iptables-restore: exit status 4 (Another app is currently holdi
ng the xtables lock. Perhaps you want to use the -w option?)
```

Cause

1. When writing iptables rules to the kernel, iptables commands (such as `iptables-restore`) will use a file lock for sync to avoid concurrent writes of multiple instances. When `iptables-restore` is executed, it tries getting a file lock or exits if the lock is held by another process.
2. The error is a soft error, and kube-proxy will try getting the lock again in the next sync cycle (or when the next Service event is triggered). If the lock cannot be obtained after several attempts, a high latency will occur during rule sync.
3. The `iptables-restore` on later versions provide a `-w(--wait)` option. If `-w=5` , `iptables-restore` will be blocked for five seconds when getting the lock. If another process releases the lock during this period, `iptables-restore` can continue its operation.

Fix guide

1. If kube-proxy is a binary deployment on the node, you can upgrade `iptables-restore` by upgrading the node OS. Below is the sample logic:

Node OS	Target Version
CentOS	7.2 or later
Ubuntu	20.04 or later
Tencent Linux	2.4 or later

2. If kube-proxy is a DaemonSet deployment in the cluster, you can upgrade `iptables-restore` by upgrading kube-proxy. Below is the sample logic:

TKE Cluster Version	Fix Policy
> 1.12	No fixes are required, as the problem doesn't exist.

TKE Cluster Version	Fix Policy
1.12	Upgrade kube-proxy to v1.12.4-tke.31 or later.
< 1.12	Upgrade the TKE cluster.

Note：
For more information on the latest TKE versions, see TKE Kubernetes Revision Version History.

Another add-on holding the iptables lock for too long

Sample error message

```
Failed to ensure that filter chain KUBE-SERVICES exists: error creating chain "KU
BE-EXTERNAL-SERVICES": exit status 4: Another app is currently holding the xtable
s lock. Stopped waiting after 5s.
```

Cause

1. When writing iptables rules to the kernel, iptables commands (such as `iptables-restore`) will use a file lock for sync to avoid concurrent writes of multiple instances. When `iptables-restore` is executed, it tries getting a file lock. If the lock is held by another process, `iptables-restore` will be blocked for a certain period of time (subject to the `-w` value, which is five seconds by default) before getting the lock. It will continue after getting the lock or exit.
2. The error indicates that the iptables file lock is held by another add-on for more than five seconds.

Fix guide

Reduce the time when other add-ons hold the iptables file lock as much as possible. In particular, the NetworkPolicy (kube-router) add-on provided on the add-on management page in the TKE console on an earlier version holds the iptables lock for a long time. You can upgrade it to the latest version `v1.3.2` .

kube-proxy to kube-apiserver connection exception

Sample error message

```
Failed to list *core.Endpoints: Stream error http2.StreamError{StreamID:0xea1, Co
de:0x2, Cause:error(nil)} when reading response body, may be caused by closed con
```

```
nection. Please retry.
```

Cause

There is a bug when Kubernetes on an earlier version calls the go HTTP/2 package, which causes the client to use a disabled connection of the API server. When this bug occurs in kube-proxy, rule sync will fail. For more information, see (1.17) Kubelet won't reconnect to Apiserver after NIC failure (use of closed network connection) #87615 and Enables HTTP/2 health check #95981.

Fix guide

Upgrade kube-proxy. Below is the sample logic:

TKE Cluster Version	Fix Policy
> 1.18	No fixes are required, as the problem doesn't exist.
1.18	Upgrade kube-proxy to v1.18.4-tke.26 or later.
< 1.18	Upgrade the TKE cluster.

> Note：
> For more information on the latest TKE versions, see TKE Kubernetes Revision Version History.

kube-proxy panicked after the first startup and became normal after restart

Sample error message

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x50 pc=0x1514fb8]
```

Cause

1. The community code of kube-proxy has a bug, where the kernel module for statistics loading is missing during initialization, leading to the use of uninitialized variables.
2. The log is not detailed enough and failed to output the result regarding whether the IPVS mode can be used. For more information, see kube-proxy panics with SIGSEGV on first run #89729, Do not forget recording loaded modules #89823, and ipvs: log err from CanUseIPVSProxier #89785.

Fix guide

Upgrade kube-proxy. Below is the sample logic:

TKE Cluster Version	Fix Policy
> 1.18	No fixes are required, as the problem doesn't exist.
1.18	Upgrade kube-proxy to v1.18.4-tke.26 or later.
< 1.18	No fixes are required, as the problem doesn't exist.

Note：
For more information on the latest TKE versions, see TKE Kubernetes Revision Version History.

kube-proxy kept panicking

Sample error message

```
Observed a panic: "slice bounds out of range" (runtime error: slice bounds out of range)
```

Cause

There is a bug in the community code of kube-proxy. When `iptables-save` is executed, the standard output and standard error are targeted at the same buffer, and the sequence of the two is uncertain, leading to an unexpected data format in the buffer and thereby a panic during processing. For more information, see kube-proxy panics when parsing iptables-save output #78443 and Fix panic in kube-proxy when iptables-save prints to stderr #78428.

Fix guide

Upgrade kube-proxy. Below is the sample logic:

TKE Cluster Version	Fix Policy
> 1.14	No fixes are required, as the problem doesn't exist.
1.14	Upgrade kube-proxy to v1.14.3-tke.27 or later.
1.12	Upgrade kube-proxy to v1.12.4-tke.31 or later.

TKE Cluster Version	Fix Policy
< 1.12	No fixes are required, as the problem doesn't exist.

Note：
For more information on the latest TKE versions, see TKE Kubernetes Revision Version History.

kube-proxy occupied high CPU periodically in IPVS mode

Cause

This is because kube-proxy frequently refreshes the node Service forwarding rules, specifically:

- kube-proxy frequently performs periodic rule syncs.
- The business Service or Pod is frequently changed.

Fix guide

If the problem is caused by frequent periodic rule syncs by kube-proxy, you need to modify relevant parameters. Below are default parameters of kube-proxy on an earlier version:

```
--ipvs-min-sync-period=1s (minimum refresh interval of one second)
--ipvs-sync-period=5s (periodic refresh every five seconds)
```

Therefore, kube-proxy refreshes the node iptables rules once every five seconds, consuming many CPU resources. You can change the configuration to:

```
--ipvs-min-sync-period=0s (real-time refresh upon event occurrence)
--ipvs-sync-period=30s (periodic refresh every 30 seconds)
```

The above configured values are default values and can be configured as needed.

Cluster API Server Inaccessibility Troubleshooting

Last updated：2022-11-02 11:43:23

Inaccessibility After Private Network Access Is Enabled

You can enable private network access in the TKE console. If resources still cannot be accessed, check the following based on your cluster type:

Managed cluster

Check whether the security group of the node in the cluster correctly opens the port range of 30000–32768 as instructed in "Viewing node security group configurations".

Self-deployed cluster

1. Check whether the security group of the node in the cluster correctly opens the port range of 30000–32768 as instructed in "Viewing node security group configurations".

- When enabling private network access, you set the VPC subnet IP range in the console. Check whether the Master node in the cluster allows this VPC subnet IP range.
- Check whether the security group of the Master node in the cluster correctly opens the VPC IP range and VPC subnet IP range where the Master node is located.

Inaccessibility After Public Network Access Is Enabled

You can enable public network access in the TKE console. If resources still cannot be accessed, check the following based on your cluster type:

Managed cluster

Check whether the source CIDR block of the security group is configured correctly. You can also set the source `0.0.0.0/0` to be fully open to the public network, and test the Internet access again.

Self-deployed cluster

When public network access is enabled for the self-deployed cluster, the `default/kubelb-internet` Service object will be automatically created in the cluster. This Service will be automatically bound to a public network CLB

instance. By default, this CLB instance will not be bound to a security group (that is, fully open to the internet), and the `EXTERNAL-IP` field shows the VIP of the CLB instance.

```
$ kubectl get service kubelb-internet
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubelb-internet LoadBalancer 172.16.252.94 152.136.8.98 443:32750/TCP 3m4s
```

1. Check whether the CLB bound to the `default/kubelb-internet` Service object has a security group configured correctly.

- Check whether the security group of the master node in the cluster correctly opens the port range of 30000–32768 as instructed in "Viewing node security group configurations".
- Check whether the security group of the Master node in the cluster correctly opens the VPC IP range and VPC subnet IP range where the Master node is located.

Service and Ingress Inaccessibility Troubleshooting

Last updated：2022-12-27 15:36:26

Inaccessibility of the Public Network or Private Network Service Provided by a Service

For a Service that has enabled provides public network or private network access, if the access fails or the CLB port status is abnormal, please check the following:

1. Check whether the security group of the node in the cluster correctly opens the port range of 30000–32768 as instructed in "Viewing node security group configurations".
2. For public network services, further check whether the node has public network bandwidth (only for bill-by-CVM accounts).
3. If the type of the Service is loadbalancer, you can ignore the CLB and directly check whether the NodeIP + NodePort are accessible.
4. Check whether Service can be accessed normally in the cluster.

Inaccessibility of the Public Network Service Provided by an Ingress

If Ingress that provides public network services cannot be accessed, please check the following:

1. If the request returns 504, check whether the security group of the node in the cluster opens the port range of 30000–32768 as instructed in "Viewing node security group configurations".
2. Check whether the CLB bound to the Ingress is configured with a security group that does not open port 443.
3. Check whether the Ingress backend Service can be accessed normally in the cluster.
4. If the request returns 404, check whether Ingress forwarding rules are correctly configured.

Inaccessibility of a Service in the Cluster

In the TKE cluster, mutual access between Pods is usually implemented through the DNS name `my-svc.my-namespace.svc.cluster.local` of the Service. If the Service cannot be accessed in the Pod, check the following:

1. Check whether the `spec.ports` field of the Service is correct.

2. Test whether the ClusterIP of the Service is accessible, and if so, the cluster DNS is abnormal. For detailed directions, see Cluster DNS Troubleshooting.
3. Test whether the endpoint of the Service is accessible, and if not, further check as instructed in "Pods on different nodes in the same cluster cannot access each other".
4. Check whether iptables or ipvs forwarding rules of the node where the Pod is located are complete.

Service Inaccessibility or Latency During Access to an Ingress Due to a CLB Loopback

Background

In order to shorten the linkage and improve the performance when the business in a cluster accesses a Service of the LoadBalancer type through CLB, the community edition of kube-proxy binds the CLB IP to the local dummy ENI in IPVS mode, so that the traffic short-circuits on the node instead of going through CLB and thus is directly forwarded to the endpoint locally. However, this optimization conflicts with the health check mechanism of CLB. The following offers a detailed analysis and solutions. When you use TKE, you may experience service inaccessibility or several seconds of latency during access to an Ingress due to a CLB loopback.

Issue description

CLB loopback may cause the following symptoms:

- No matter whether you are in iptables or IPVS mode, when you access an Ingress over the cluster private network, a 4-second latency or inaccessibility occurs.
- In IPVS mode, when you access a private network Service of the LoadBanacer type in your cluster, it is completely inaccessible, or the connection is unstable.

Workaround

- Avoiding a L4 loopback
- Avoiding a L7 loopback

> Note
> CLB is required to support the non-VIP health check source IP to avoid a L4 loopback. This feature is currently in beta test. To try it out, submit a ticket for application.

To solve the loopback problem that may be encountered when using the Service, follow the steps below:

1. Check whether kube-proxy is on the latest version, and if not, upgrade it as follows:
   i. To solve this problem, kube-proxy needs to support binding the LB address to the IPVS ENI. You can find version IDs that support this capability in TKE Kubernetes Revision Version History, such as v1.20.6-tke.12, v1.18.4-tke.20, v1.16.3-tke.25, v1.14.3-tke.24, and v1.12.4-tke.30. Later versions also support this capability.

2022-01-20	v1.20.6-tke.12	• EKS rescheduling optimization: Lower the score for super nodes that have been drained in the same availability zone. (kube-scheduler) • The apiserver supports integration of ExternalName type external services. (kube-apiserver) • Supports binding the LB addresses to the ipvs ENIs. (kube-proxy)

   ii. Determine the cluster version: You can log in to the TKE console and view the version ID of the current cluster on the **Basic information** page in the cluster. The following shows a cluster on v1.22.5:

Kubernetes version	Master 1.22.5-tke.5(Latest version) Node 1.22.5-tke.5

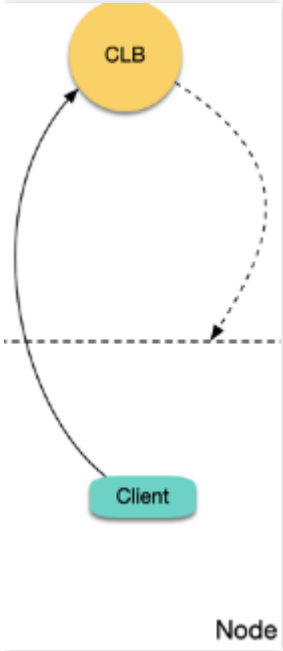
   iii. Find the DaemonSet named kube-proxy under the kube-system namespace, update the version number of its image, and select a version that supports this capability or a later version. For example, for a cluster on v1.18, you need to select an image version later than v1.18.4-tke.20.
2. Annotate all Services:
   `service.cloud.tencent.com/prevent-loopback: "true"`
   Output image:

- kube-proxy will bind the CLB VIP to the local system based on this information to solve the loopback problem.
- The service-controller will call the CLB API and change its health check IP to an IP in the 100.64 IP range to solve the health check problem.

Causes

The root cause is that when CLB forwards a request to the real server, the packet source and destination IPs are both on the same node, but Linux will ignore the received packets whose source IP is the local IP by default, causing the

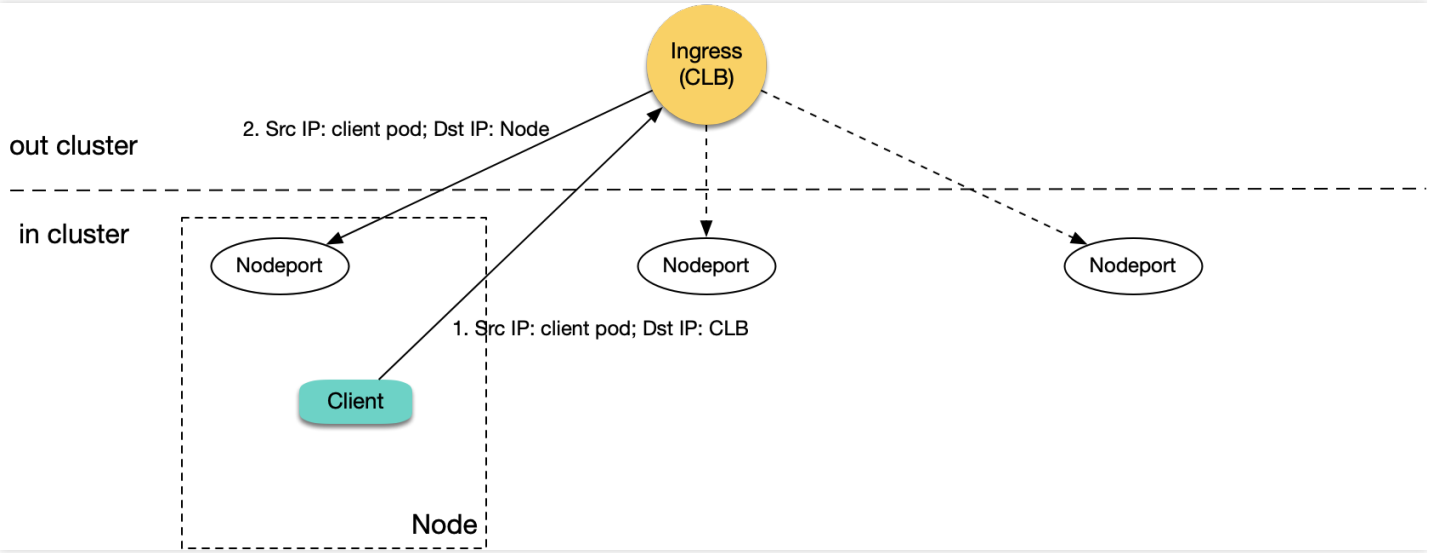
data packet to be looped within the CVM instance, as shown below:



- Analyzing a L7 loopback
- Analyzing a L4 loopback

If you use a TKE CLB Ingress, a CLB instance and layer-7 listener rules (HTTP/HTTPS) for ports 80 and 443 will be created for each Ingress resource, and the same NodePort of each corresponding TKE node will be bound to each Ingress location as the real server (a location corresponds to a Service, and each Service uses the same NodePort of each node to expose the traffic). CLB forwards the request to the corresponding real server (i.e., NodePort) according to the location matched by the request, and the traffic will be forwarded to the corresponding backend Pod after passing the NodePort and the Kubernetes iptables or IPVS. When a Pod in the cluster accesses the private network

Ingress in the cluster, CLB will forward the request to the corresponding NodePort of a node in the cluster.



As shown above, when the node whose traffic is forwarded is the node of the client that sends the request:

1. A Pod in the cluster accesses CLB, and CLB forwards the request to the corresponding NodePort of any node.
2. When the packet reaches the NodePort, the target IP is the node IP, and the source IP is the actual IP of the client Pod. As CLB does not perform SNAT, it will pass through the actual source IP.
3. As the source and target IPs are both on the same server, loopback will occur, and CLB cannot receive response from the real server.

The most frequent fault of access to an Ingress in the cluster is a latency of several seconds. It is because if a layer-7 CLB instance's request to a real server times out (in about 4 seconds), the instance will try the next real server. Therefore, if you set a long timeout period on the client, loopback may occur with a symptom of slow request response with a several-second latency. If your cluster has only one node, CLB has no real server for retry, and the symptom will be inaccessibility.

FAQs

Why public network CLB does not have a loopback?

There is no loopback issue if you use a public network Ingress or public network LoadBalancer Service. This is because the source IP of the packets received by public network CLB is the public network egress IP of a CVM instance, but the CVM instance cannot sense its own public network IP internally. When a packet is forwarded to the CVM instance, the public network source IP will not be considered as the local server IP, so there will not be loopback.

Does CLB have any mechanism to avoid a loopback?

Yes. CLB will determine the source IP and will not consider forwarding the request to the real server with the same IP; instead, it will forward the request to another real server. However, the source Pod IP is different from the real server

IP, and CLB does not know whether the two IPs are on the same node, so the request may still be forwarded and cause loopback.

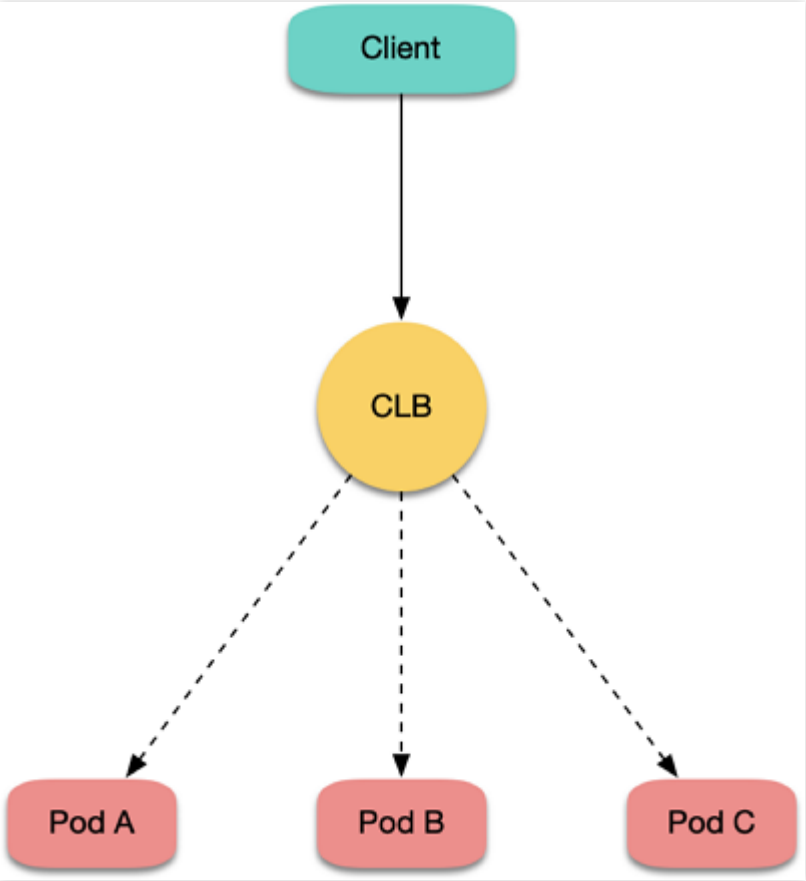
Can anti-affinity deployment of the client and server avoid a loopback?

If you deploy a client and a server through anti-affinity so that they are not on the same node, can CLB loopback be avoided?

By default, LB is bound to a real server through a node NodePort, and a request may be forwarded to NodePort of any node. In this case, no matter whether the client and server are on the same node, loopback may occur. However, if `externalTrafficPolicy: Local` is set for the Service, LB will forward them to only the node with a Server Pod. If the client and server are scheduled to different nodes through anti-affinity, no loopback will occur. Therefore, anti-affinity and `externalTrafficPolicy: Local` can avoid the loopback issue (including private network Ingress and private network LoadBalancer Service).

Does the direct LB-Pod connection of VPC-CNI have the CLB loopback issue?

TKE generally uses the Global Router network mode, and another mode is VPC-CNI (elastic network interface). Currently, direct LB-Pod connection supports only the Pods of VPC-CNI; that is, LB is directly bound to the backend Pod rather than NodePort as the real server as shown below:



In this case, requests can bypass the NodePort instead of possibly being forwarded to any node like before. However, if the client and server are on the same node, loopback may still occur, which can be avoided through anti-affinity.

Are there any suggestions?

The anti-affinity and `externalTrafficPolicy: Local` methods are not very graceful. Generally, you should avoid accessing the CLB instance in the cluster for a service in the cluster, as the service is already in the cluster, and forwarding through CLB not only lengthens the network linkage but also may cause loopback.

When you access a service in the cluster, use the service name such as `server.prod.svc.cluster.local` as much as possible. In this way, requests will not pass CLB, and loopback will not be caused.

If your business has a coupled domain name and cannot use a Service name, you can use the rewrite plugin of CoreDNS to point the domain name to a Service in the cluster. Below is the sample CoreDNS configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
name: coredns
namespace: kube-system
data:
Corefile: |2-
.:53 {
rewrite name roc.example.com server.prod.svc.cluster.local
...
```

If multiple Services share the same domain name, you can deploy an Ingress Controller (such as nginx-ingress) by yourself:

1. Use the aforementioned rewrite method to point the domain name to the self-built Ingress Controller.
2. Match the self-built Ingress with a Service according to the request location (domain name + path) and forward the request to the backend Pod. The entire linkage will not pass CLB, so loopback can be avoided.

Troubleshooting for Pod Network Inaccessibility

Last updated：2023-10-19 17:53:44

This document describes common network issues that may occur in various scenarios in TKE clusters and provides troubleshooting methods. When encountering such issues, you are advised to follow the inspection suggestions below to perform troubleshooting. If you still cannot access networks normally after confirming that the inspection items are correct, contact us for help.

Inaccessibility Between Containers (Pods) on Different Nodes in a Cluster

Pods on different nodes in the same cluster can directly access each other. If a pod on a node cannot access a pod on another node, you are advised to perform the following checks:
1. Check whether the above nodes can access each other.
2. Check whether the node security group correctly allows the container network segment and the VPC network segment or VPC subnet segment where the peer node is located.

Inaccessibility Between a Node and a Container (Pod) in the Same VPC

A node and a pod in the same VPC can directly access each other. If an inaccessibility issue occurs, you are advised to perform the following checks:
1. Check whether the peer node and the node where the pod is located can access each other.
2. Check whether the security group of the node where the pod is located correctly allows the VPC subnet segment where the peer node is located.
3. Check whether the security group of the peer node correctly allows the container segment.

Inaccessibility Between a Node and a Container (Pod) or Between Containers (Pods) in Different VPCs

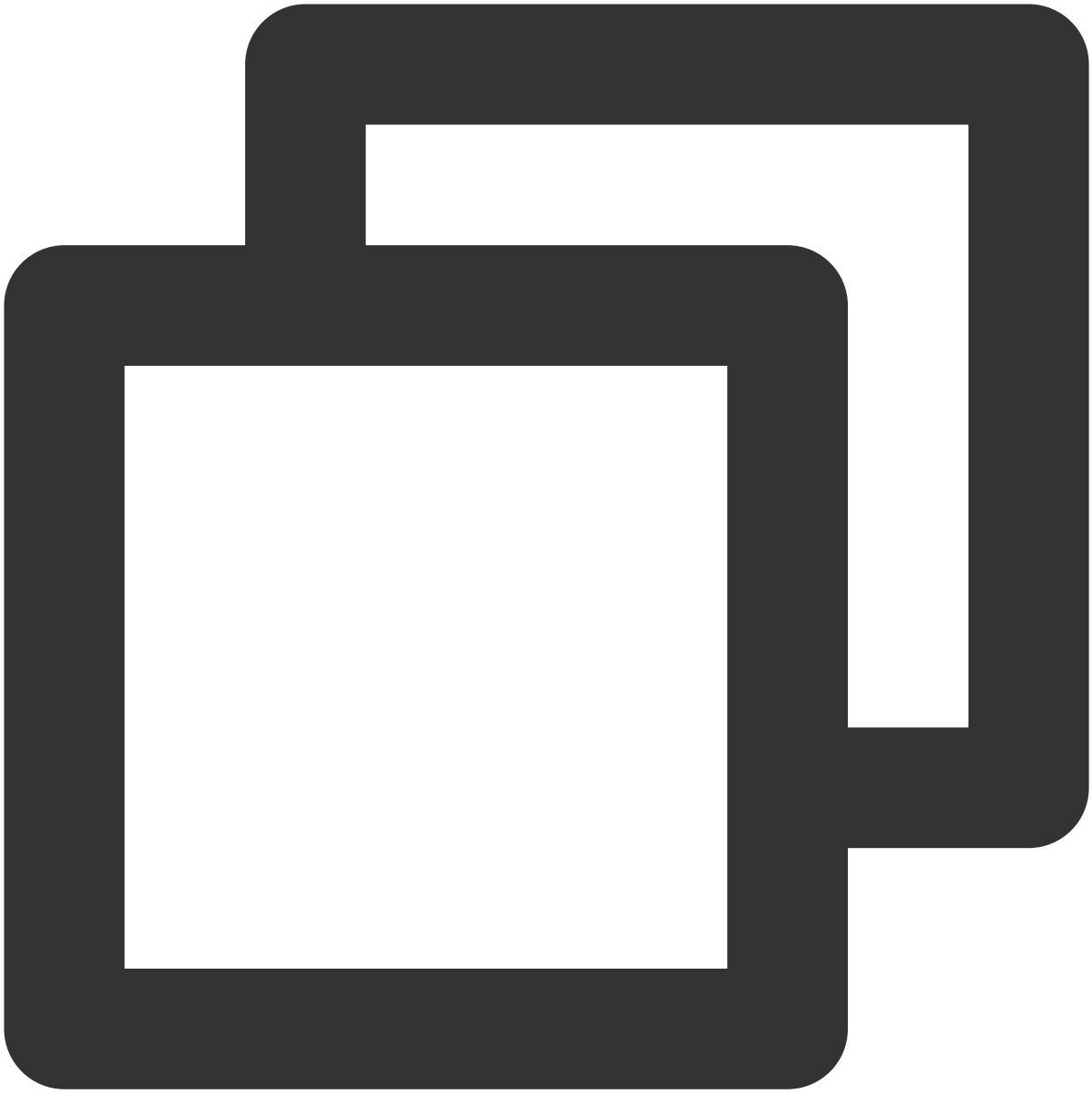
The mutual access between different VPCs must be completed through Cloud Connect Network or Peering Connection. If inaccessibility issues persist after the connection is established, you are advised to perform the

following checks:

1. Check whether the nodes can access each other.
2. Check whether the security group of the peer node correctly allows the VPC network segment and container network segment.
3. Check whether the security group of the node where the pod is located correctly allows the VPC network segment or VPC subnet segment of the peer node.

If containers (pods) cannot access each other, perform the following checks:

1. Check whether the security group of the node where the pod is located correctly allows the peer VPC network segment (or the VPC subnet segment where the node is located) and the container network segment.
2. To view the source IP address of the pod, run the following command to modify the configuration of `ip_masq_agent` and add the container network segment of each other.



```
kubectl -n kube-system edit configmap ip-masq-agent-config
```

Inaccessibility Between an IDC and a Container (Pod)

The mutual access between an IDC and a pod must be completed through Cloud Connect Network or Direct Connect Gateway. If inaccessibility issues persist after the connection is established, you are advised to perform the following checks:

Check whether the IDC firewall allows the container network segment and CVM network segment.

Check whether the CVM security group allows the IDC network segment.

Check whether the IDC uses the BGP protocol:

If the BGP protocol is not used, you need to configure the next-hop route to the direct connect gateway in the IDC for accessing the container network segment.

If the BGP protocol is used, automatic synchronization will be performed and typically no configuration is required. If the IDC has special static configurations, you can contact the O&M personnel to configure the next-hop to the direct connect gateway for accessing the container network segment.

Note

To view the IP address of a pod in an IDC, you need to allow the IDC network segment.

By default, the access to packets outside of VPCs will be converted to NodeIP through SNAT processing. When allowing an IDC network segment, you need to implement the configuration of bypassing SNAT processing.

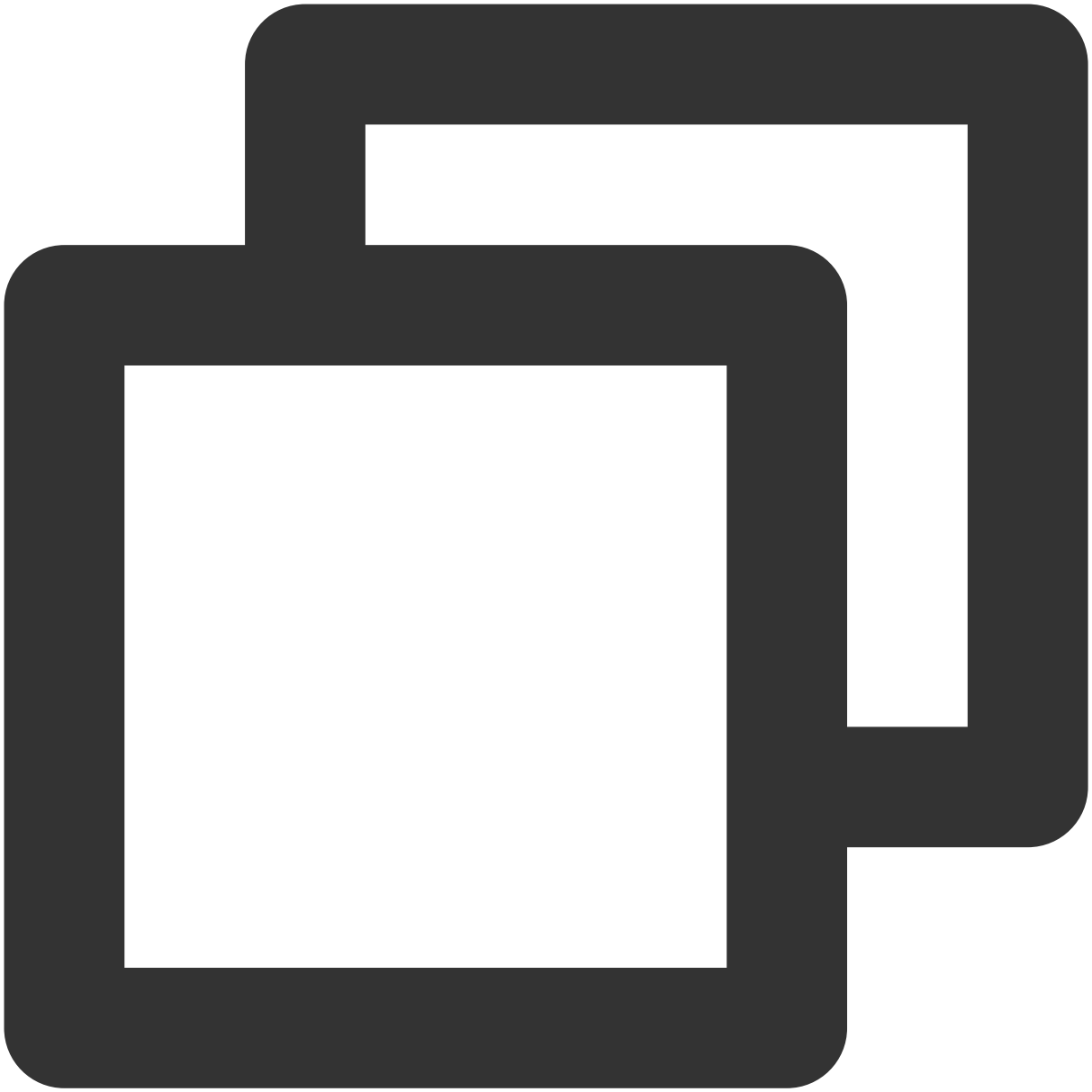
The method of allowing an IDC network segment is as follows: Run the `kubectl -n kube-system edit configmap ip-masq-agent-config` command, modify the **ip-masq-agent** configuration, and add the IDC network segment to the NonMasqueradeCIDRs list.

Related Operations

Viewing iptables or IPVS Forwarding Rules on a Node

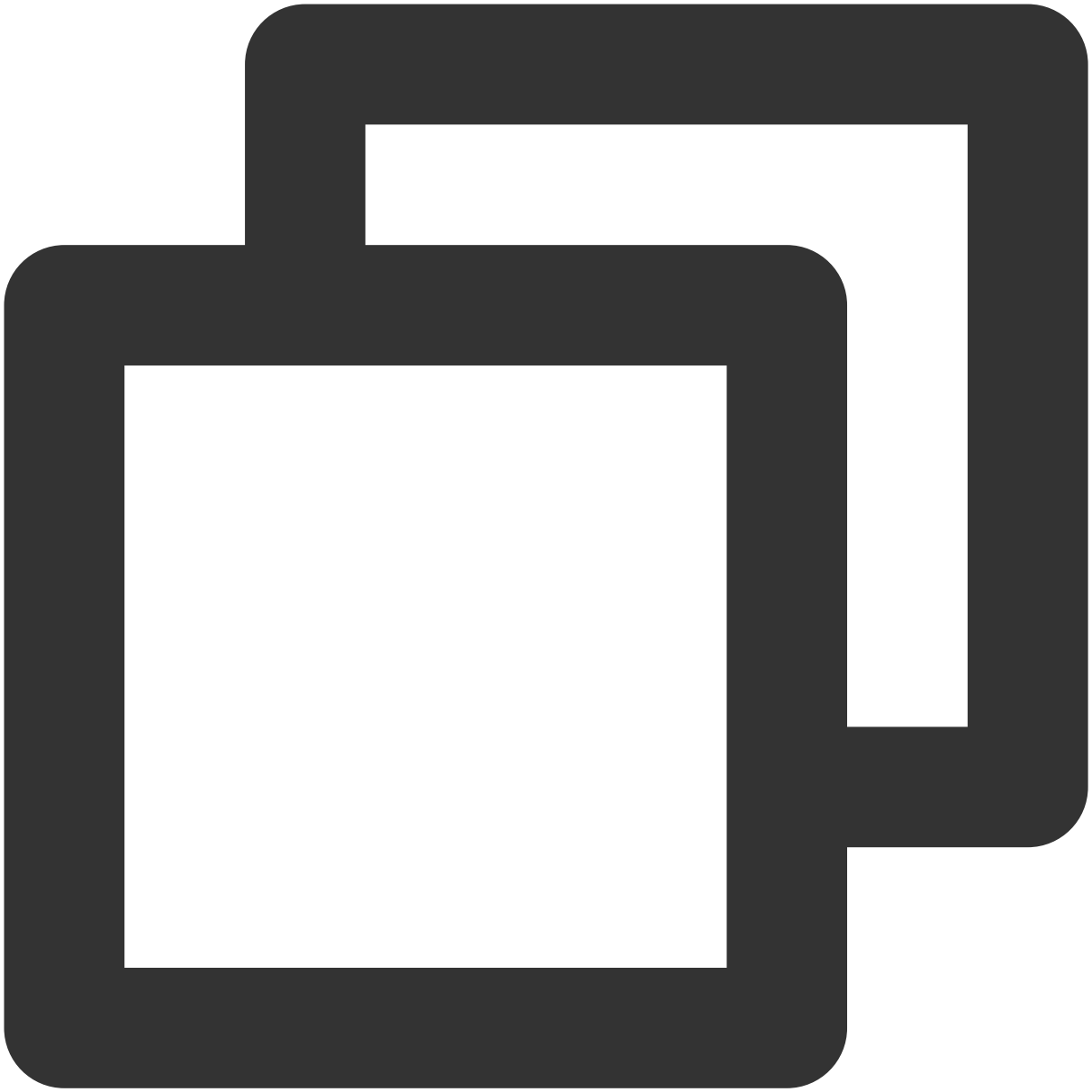
You can run the following commands to view the iptables or IPVS forwarding rules on a node.

Run the following command to view the iptables forwarding rules.



```
iptables-save
```

Run the following command to view the IPVS forwarding rules.

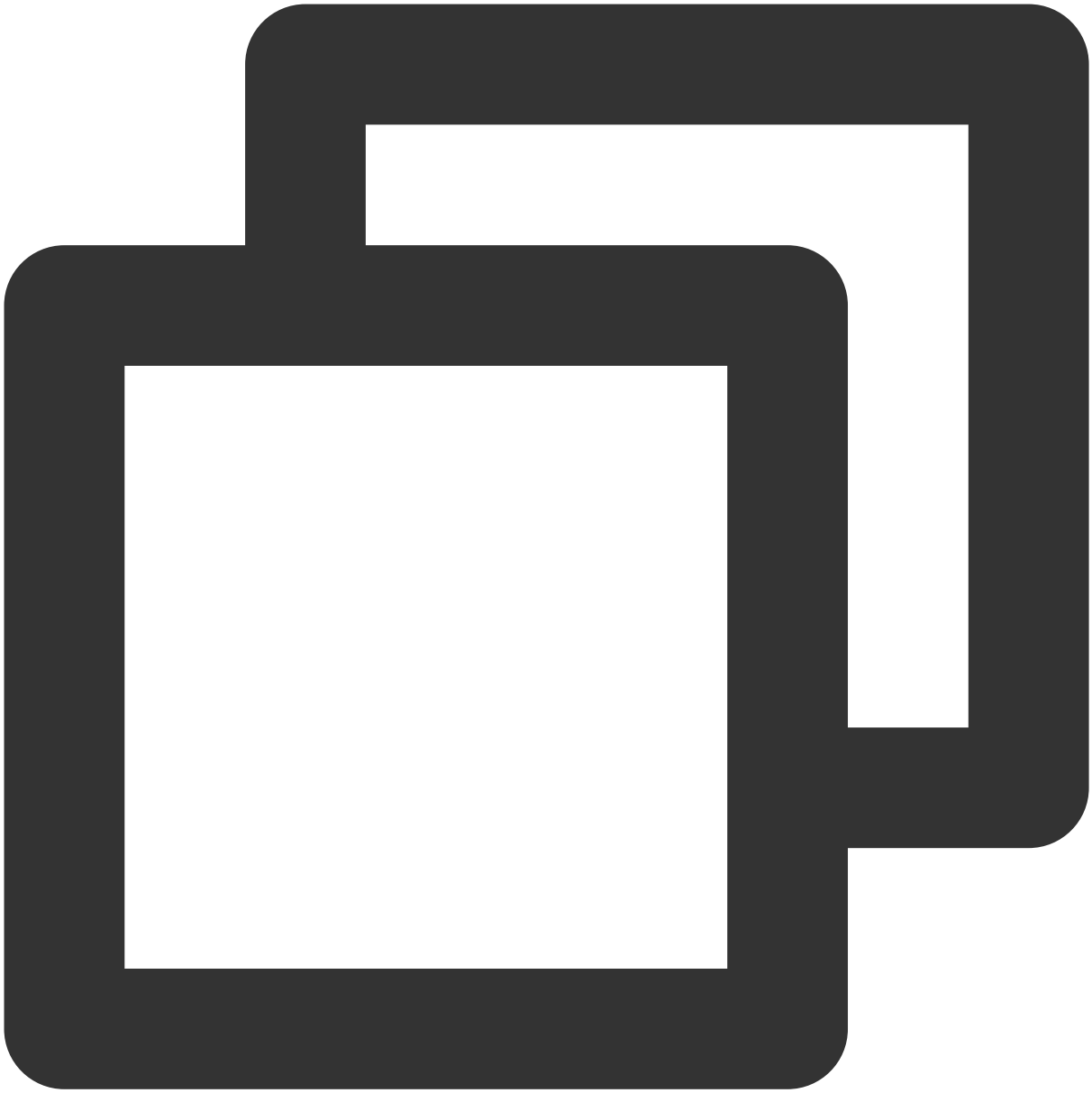


```
ipvsadm -Ln -t/-u ip:port
```

Running Packet Capture Commands

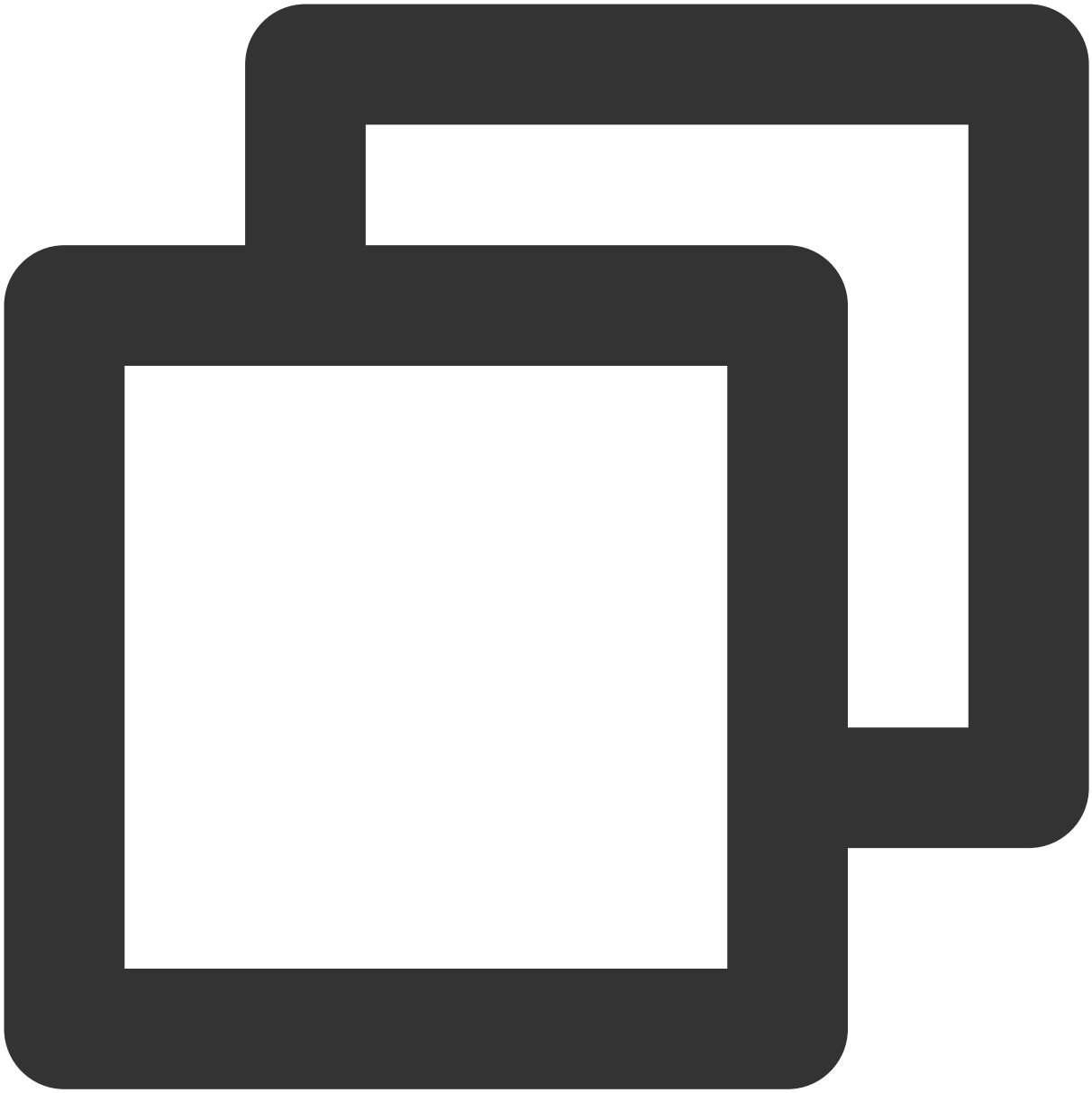
The following packet capture commands can be used to analyze situations where containers (pods) on different nodes in a cluster cannot access each other.

Run the following command to capture packets of the NIC eth0 on the node of the source pod.



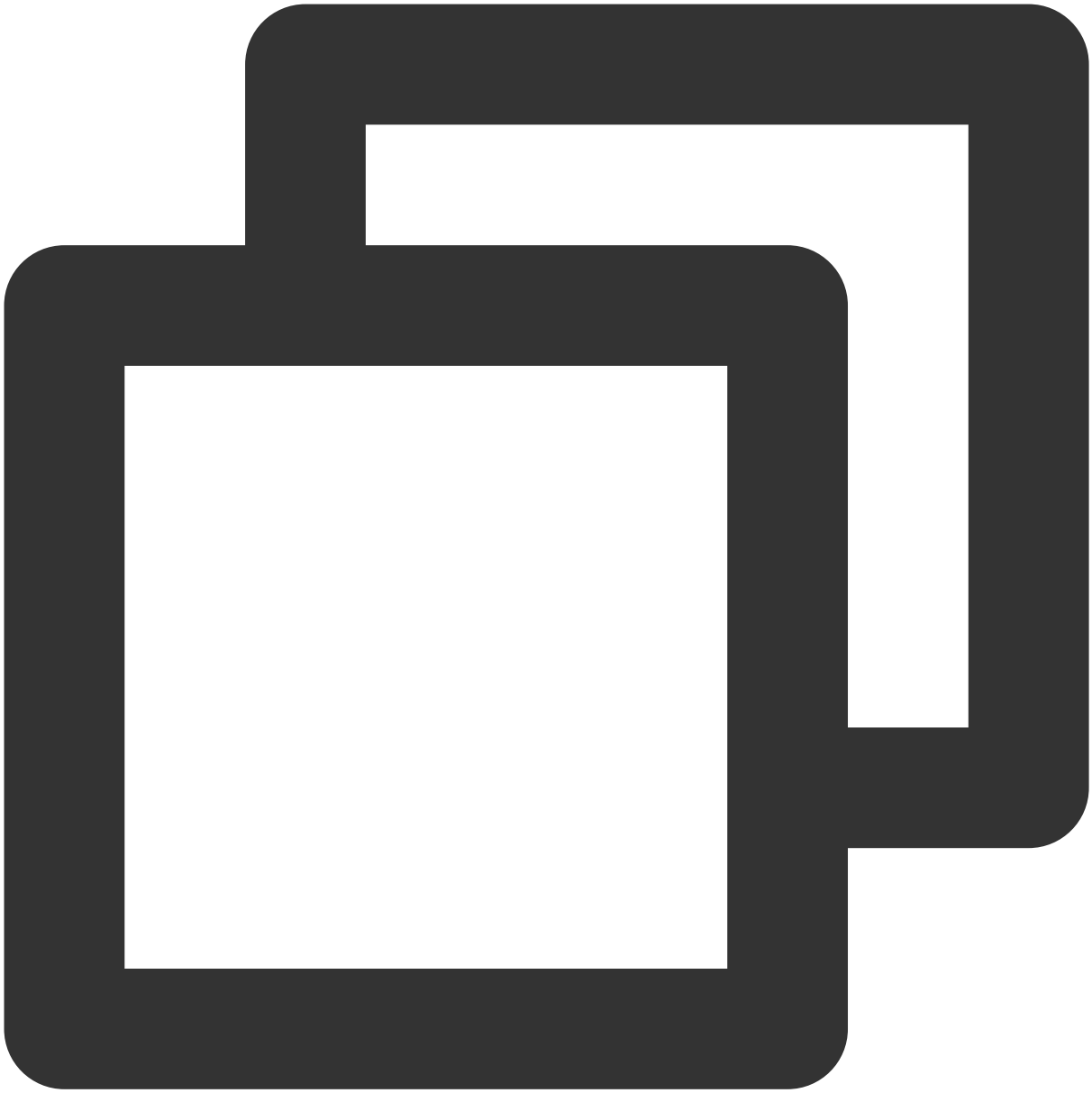
```
tcpdump -nn -vv -i eth0 host <IP address of the peer pod>
```

Run the following command to capture packets of the NIC eth0 on the node of the peer pod.



```
tcpdump -nn -vv -i eth0 host <IP address of the source pod>
```

Run the following command to capture packets of the NIC eth0 in the netns of the peer pod.



```
tcpdump -nn -vv -i eth0 port <Requested port number>
```

Locating Network Issues by Capturing Packets in a Container

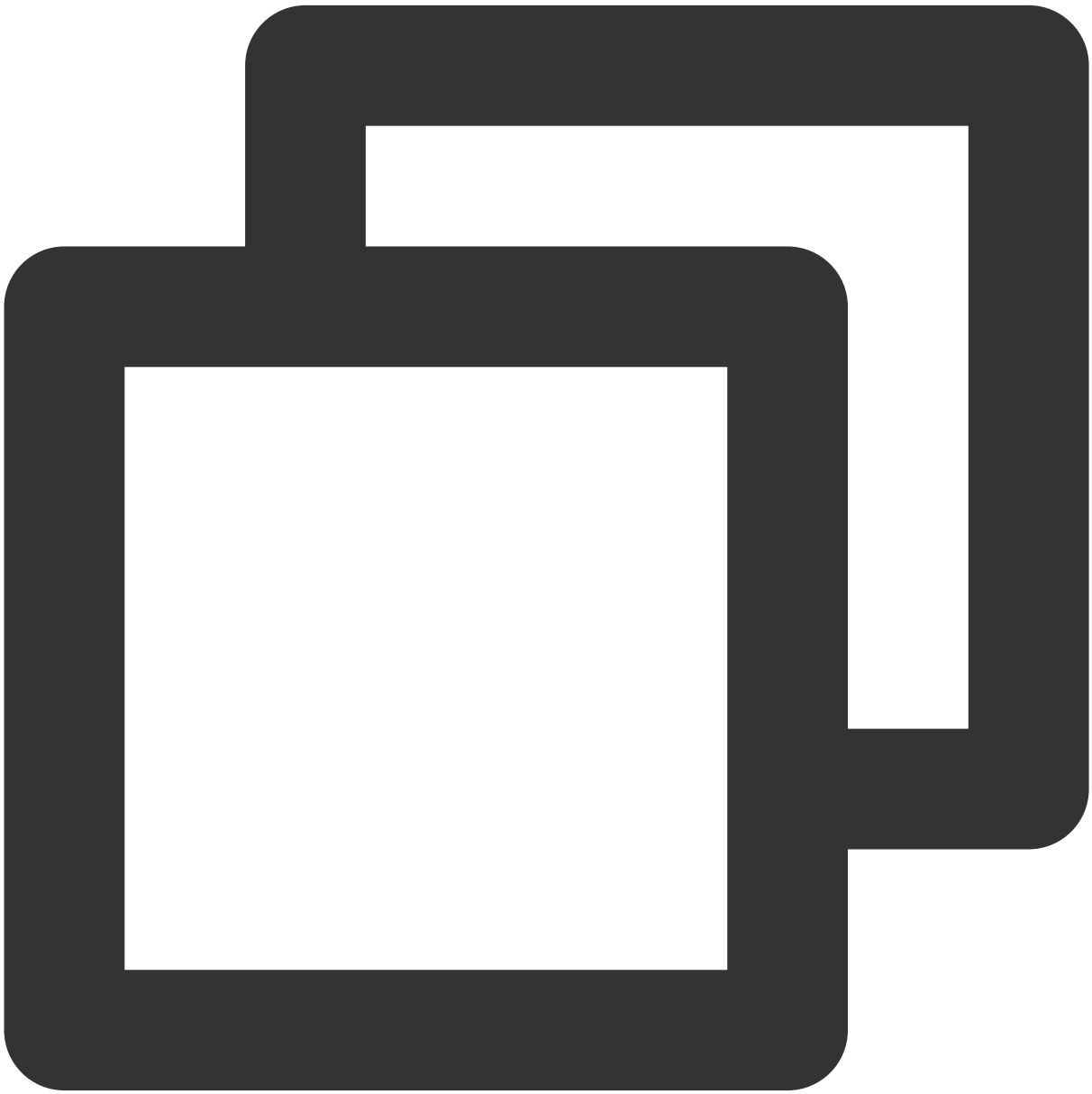
When running applications by using Kubernetes, you may encounter some network issues, the most common of which are server unresponsiveness (timeout) and abnormal packet return content. If you cannot locate an issue in the related configurations, you need to check whether the data packets are ultimately routed to the container, or whether the content of the packets arriving at and leaving the container aligns with expectations, and further narrow down the

issue scope by analyzing the packets. This topic provides a script that allows one-click access to the container network namespace (netns) and uses tcpdump on the host for packet capturing.

Using a Script to Access the Pod netns in One-Click Mode for Packet Capturing

If a service cannot be accessed, you are advised to set the number of replicas to 1 and perform the following steps to capture packets.

1. Run the following command to obtain the node where the replica is located and the pod name.



```
kubectl get pod -o wide
```

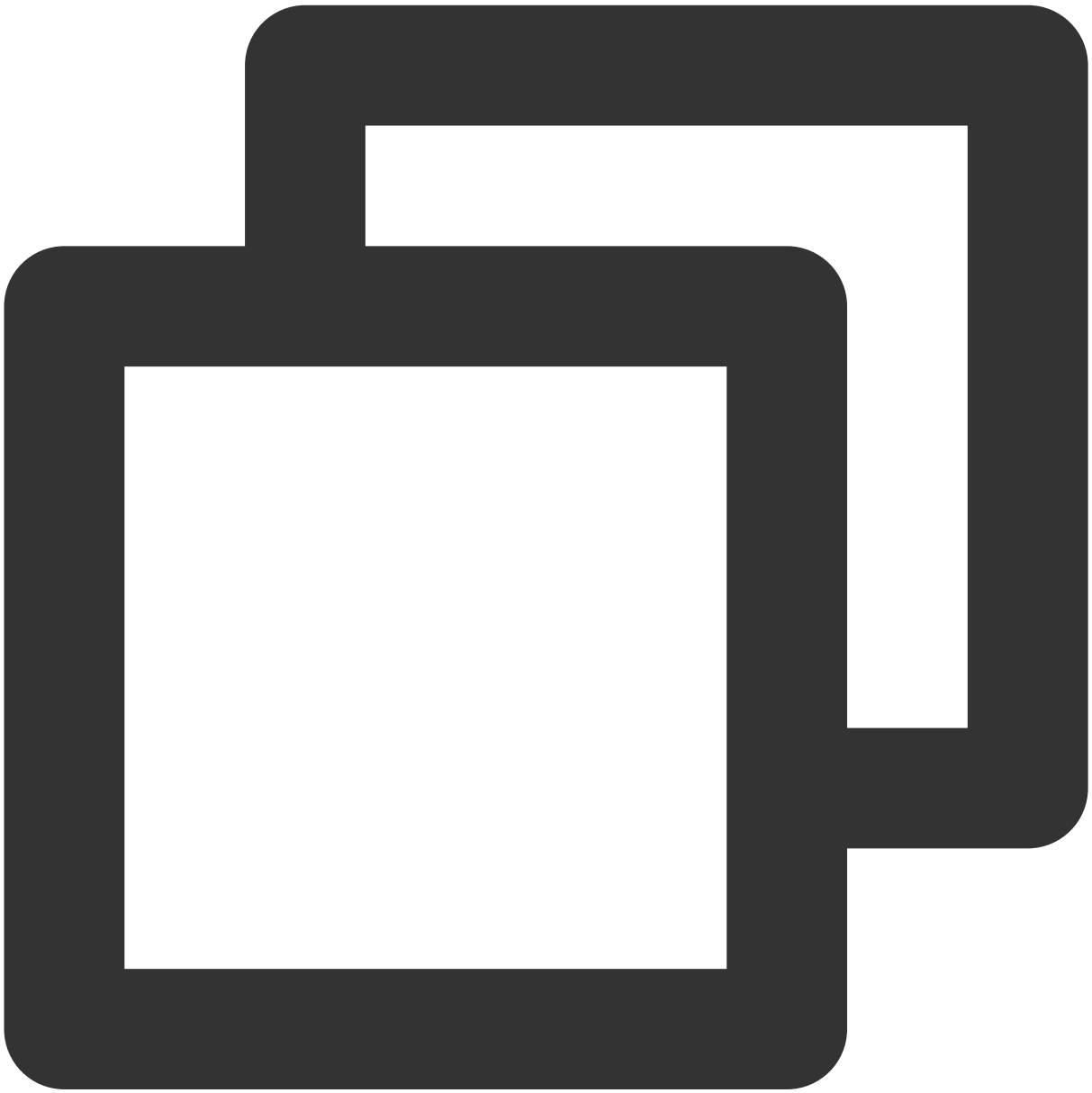
2. Log in to the node where the pod is located, and paste the following script into the Shell to register the function to the currently logged-in Shell.

```
function e() {
    set -eu
    ns=${2-"default"}
    pod=`kubectl -n $ns describe pod $1 | grep -A10 "^Containers:" | grep -Eo 'dock
    pid=`docker inspect -f {{.State.Pid}} $pod`
    echo "entering pod netns for $ns/$1"
    cmd="nsenter -n --target $pid"
    echo $cmd
```

```
    $cmd
}
```

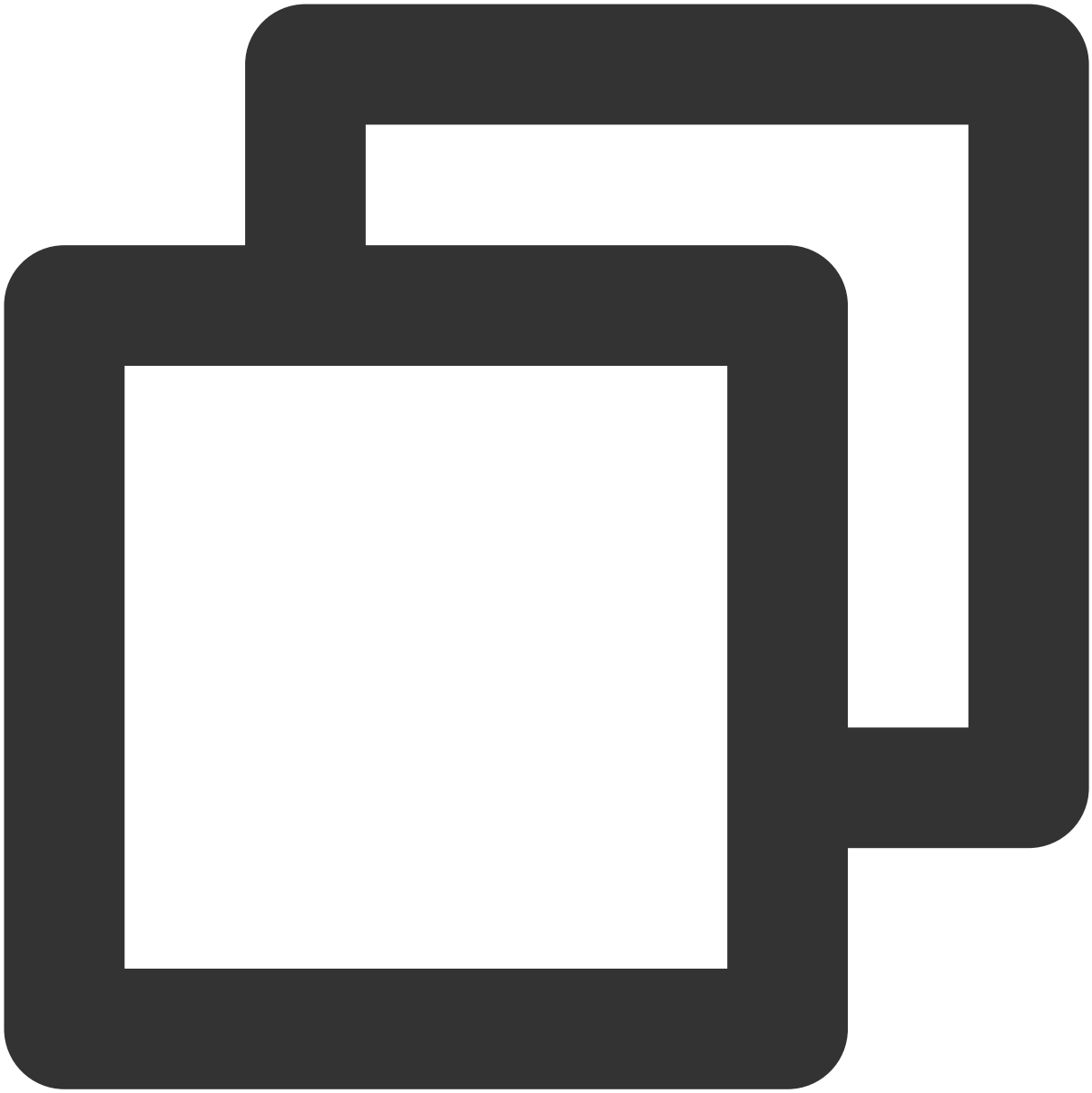
You can learn the script by referring to Script Principles and then use the script.

3. Run the following command to enter the netns where the pod is located in one-click mode.



```
e POD_NAME NAMESPACE
```

An example is as follows:

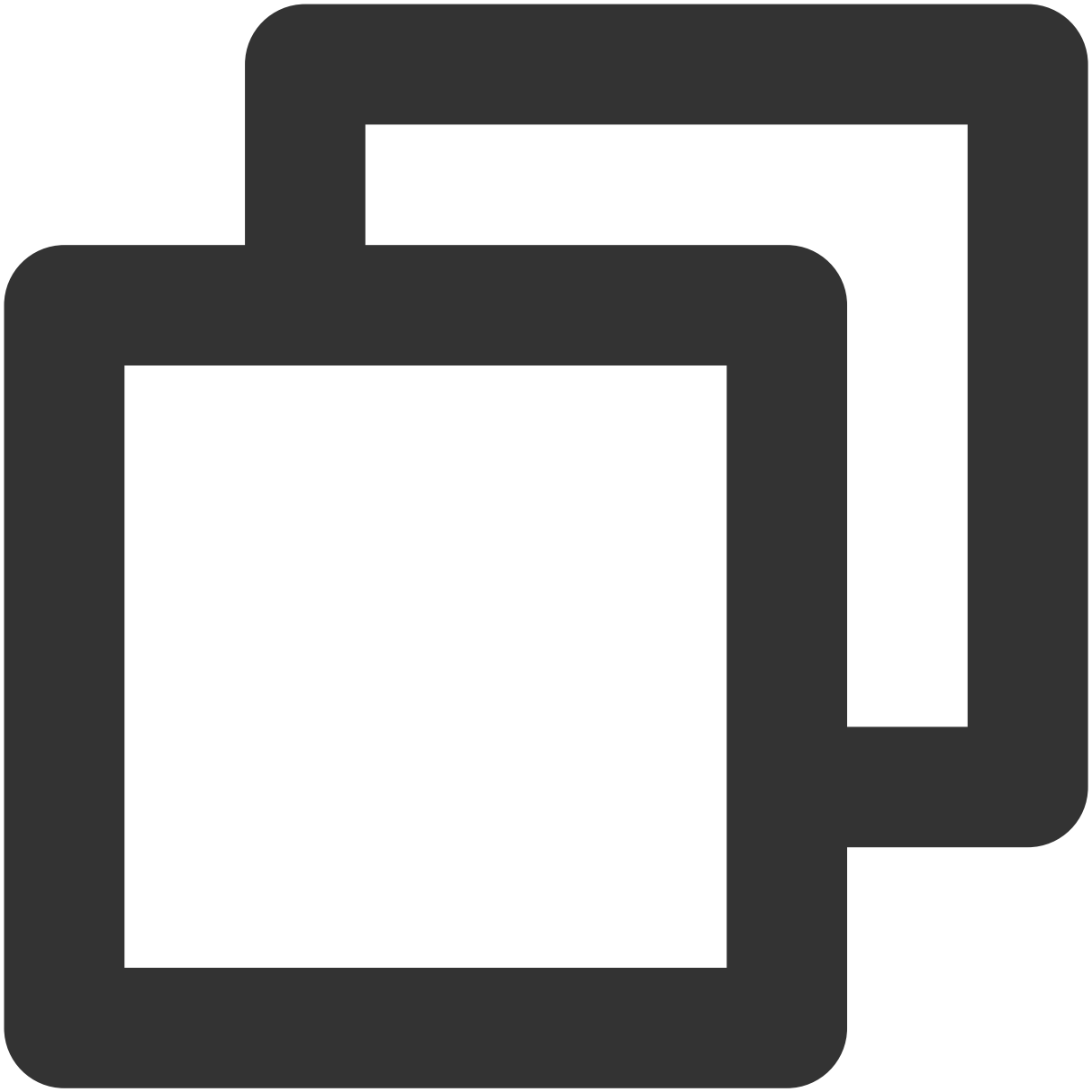


```
e istio-galley-58c7c7c646-m6568 istio-system
e proxy-5546768954-9rxg6 # Omit the namespace and use the defaut value.
```

Note

After entering the netns of the pod, you can run the `ip a` or `ifconfig` command on the host to view the NIC of the container and run the `netstat -tunlp` command to view the listening port of the current container.

4. Run the following command to capture packets using tcpdump.

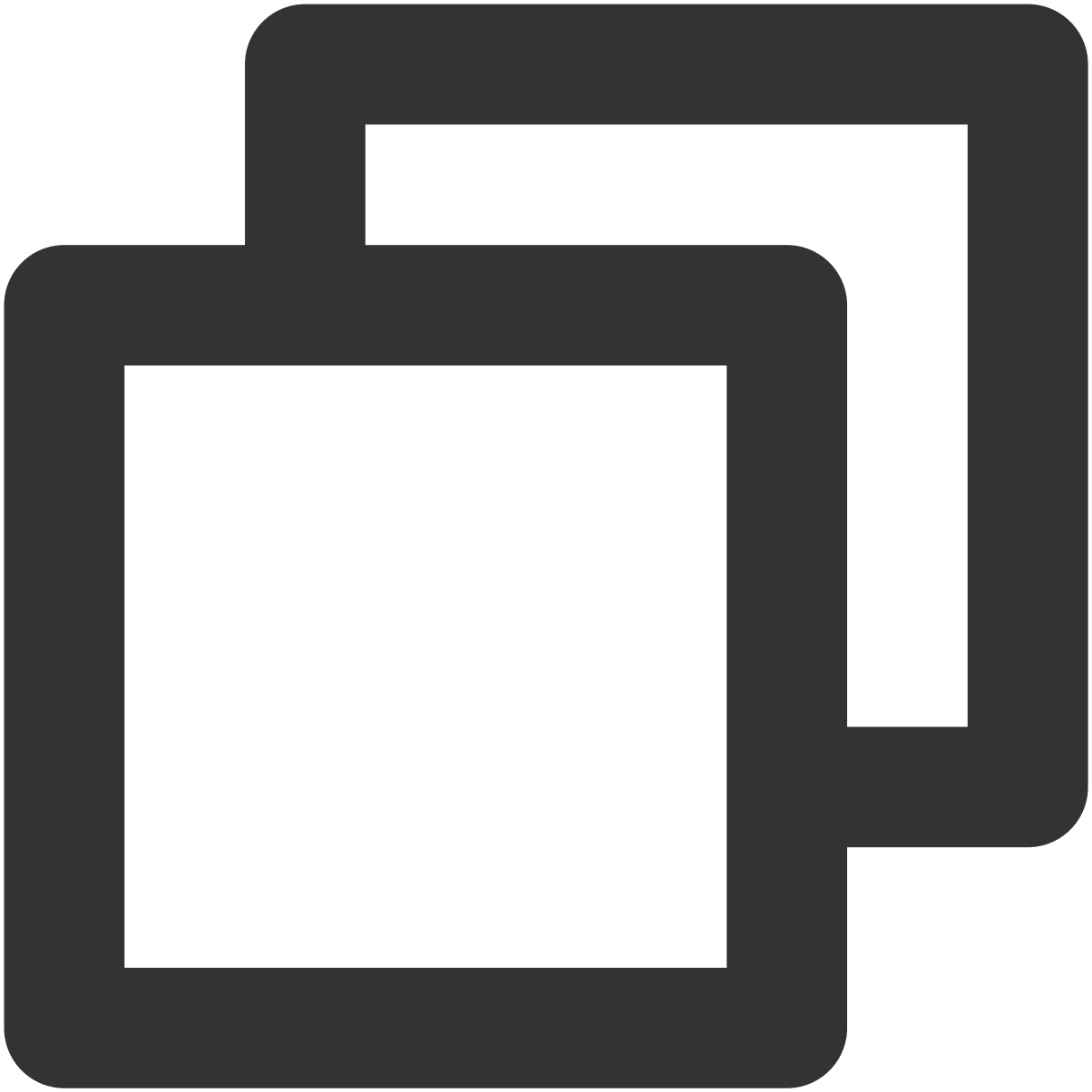


```
tcpdump -i eth0 -w test.pcap port 80
```

Analyzing Packets by Using Wireshark

You can stop packet capturing by running the `Ctrl+C` command, and then run the `scp` or `sz` command to download the captured packets for analysis by using `wireshark`. During the analysis process, you may use the following common `wireshark` filtering syntax:

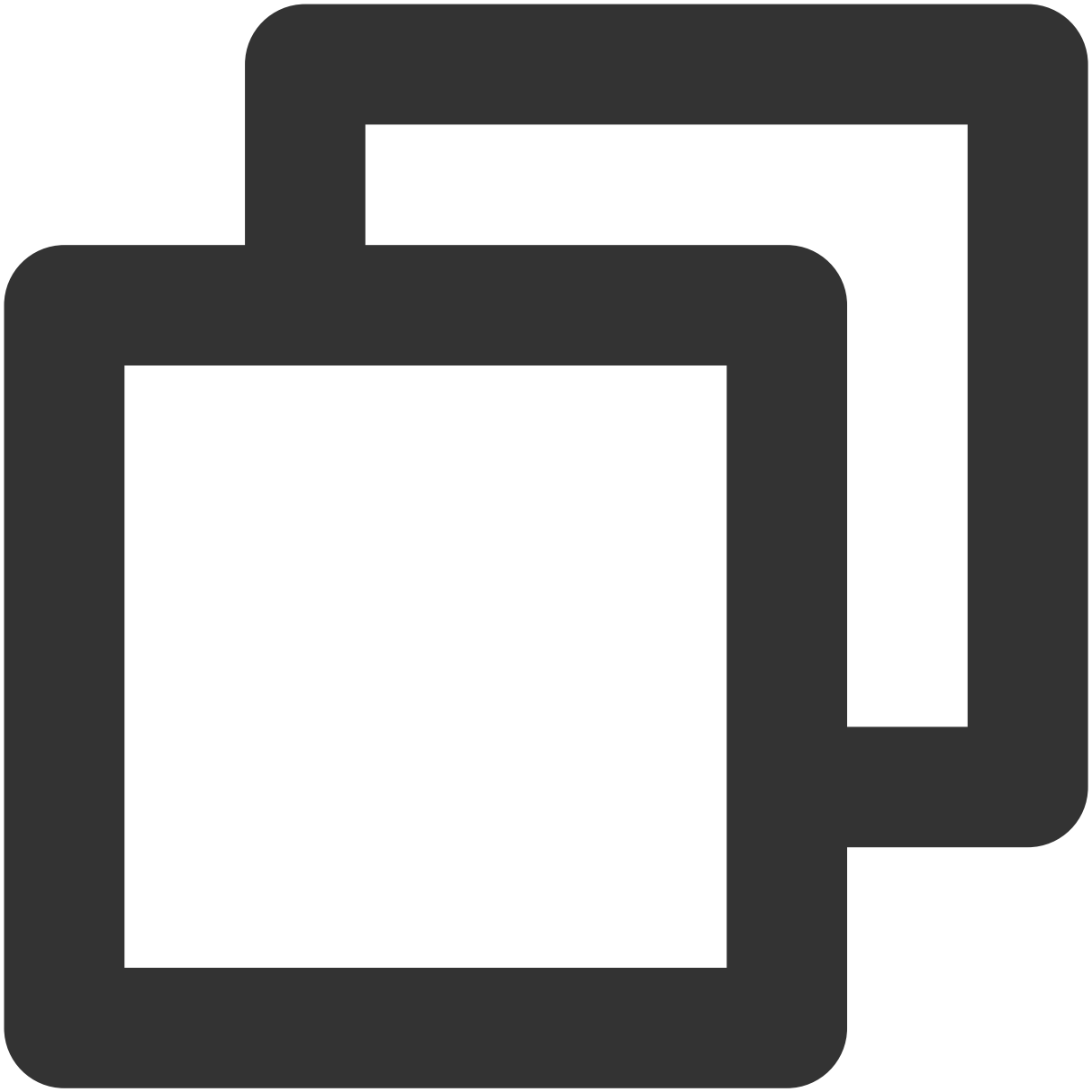
Establish a Telnet connection and send test text, such as `"lbtest"`. Run the following command to check whether the sent test packet is delivered to the container.



```
tcp contains "lbtest"
```

If the container offers the HTTP service, you can use curl to send test path requests.

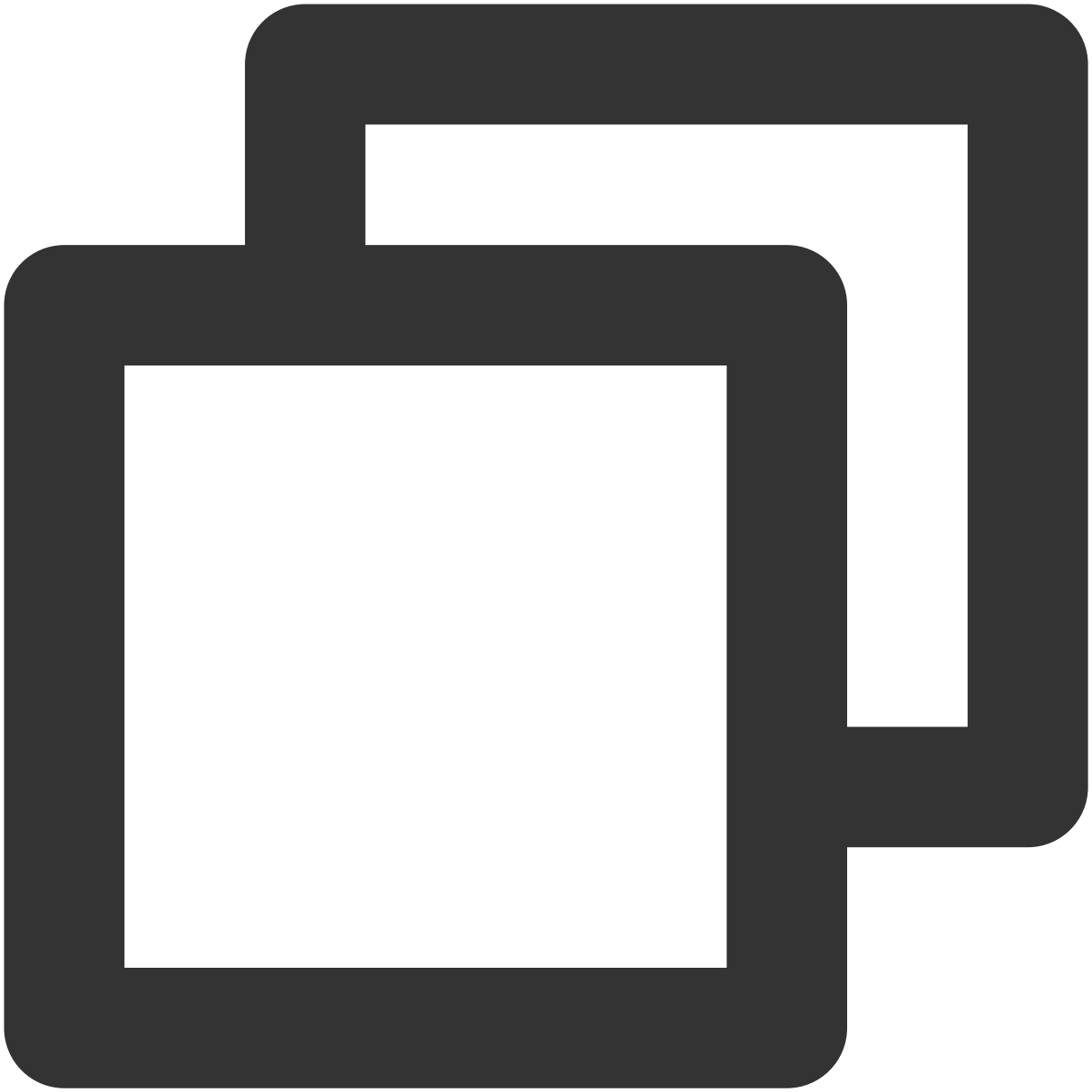
Run the following command to filter the URI, and check whether the packet is delivered to the container.



```
http.request.uri=="/mytest"
```

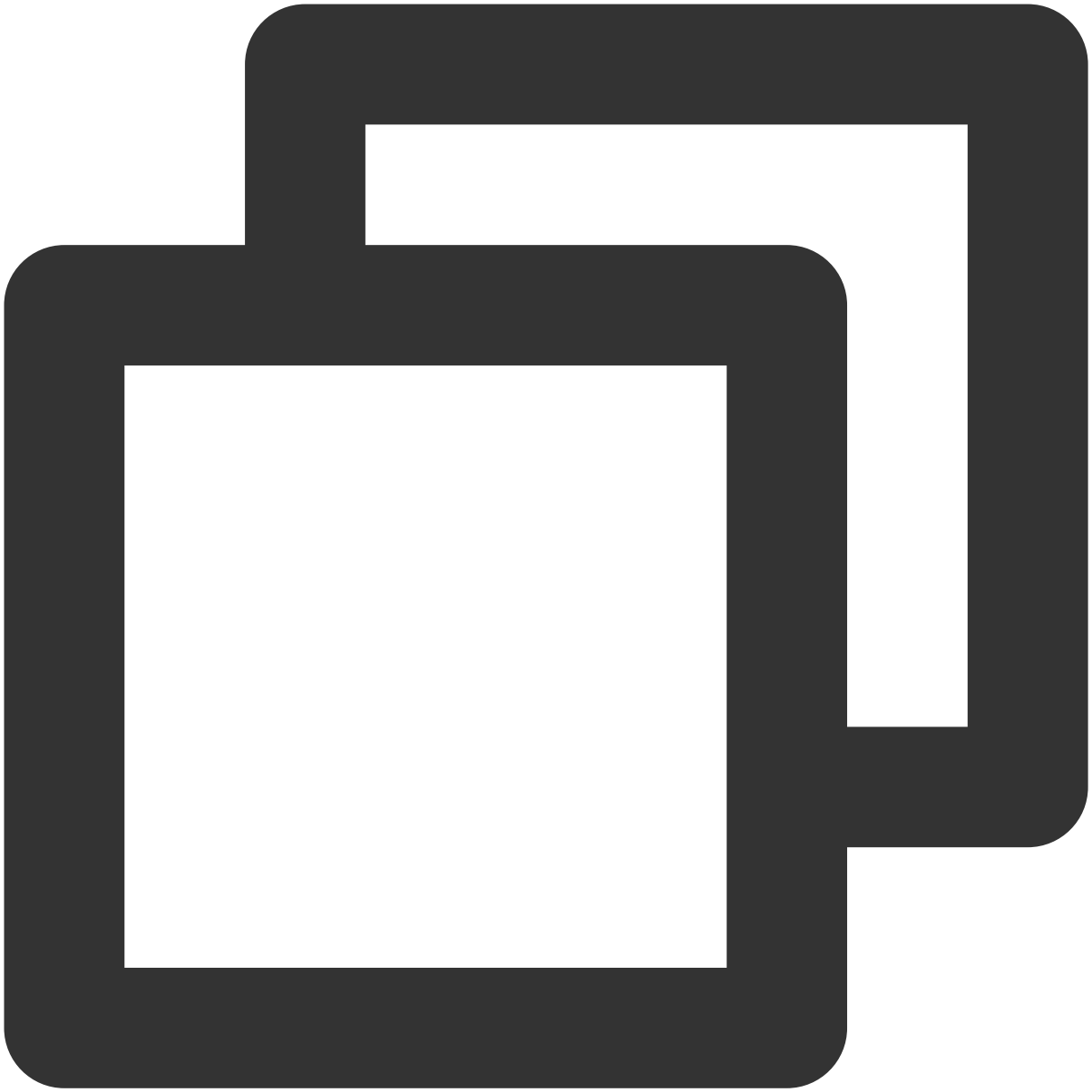
Script Principles

View the container ID for which the specified pod runs.



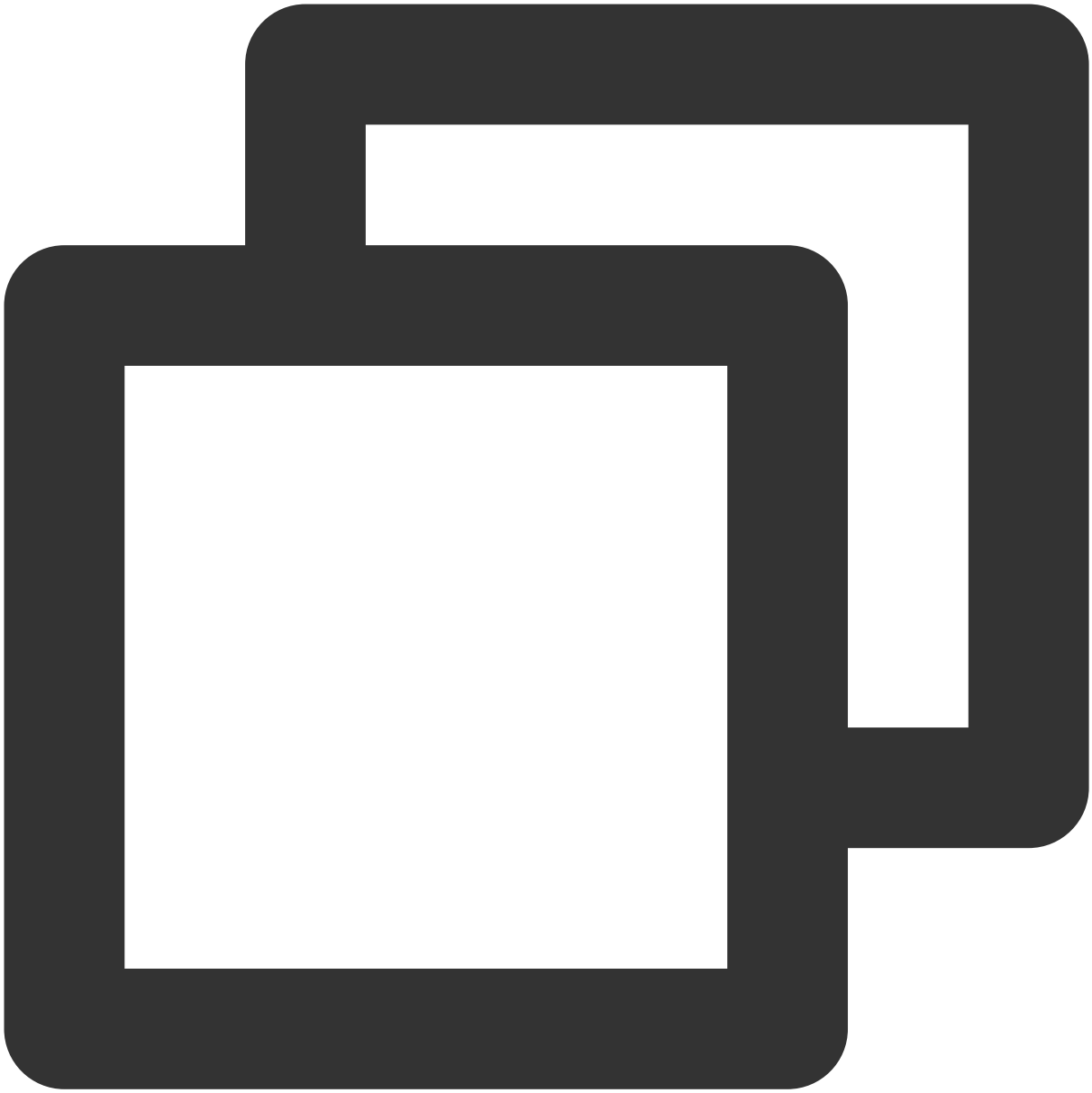
```
kubectl describe pod <pod> -n mservice
```

Obtain the PID of the container process.



```
docker inspect -f {{.State.Pid}} <container>
```

Enter the network namespace of the container.



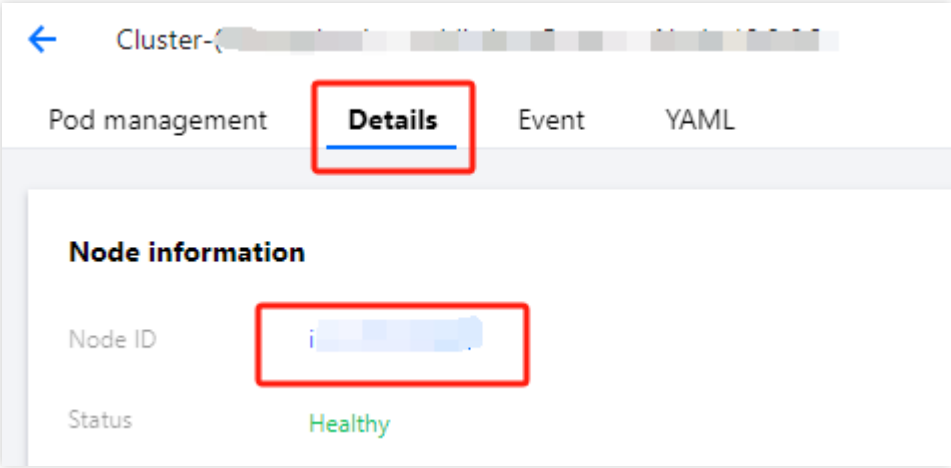
```
nsenter -n --target <PID>
```

The host names that the above script depends on include: kubectl, docker, nsenter, grep, head, and sed.

Viewing the Node Security Group Configuration

1. Log in to the TKE console and choose Cluster in the navigation bar.
2. Click the cluster ID to go to the cluster details page.
3. In the navigation bar, choose **Node management > Node**.
4. On the node list page, click the ID of the node for which you want to view the security group.

5. On the node management page, click the **Details** tab and click the node ID under **Node information**.



6. On the basic information page of the node, click the **Security group** tab, and check whether the security group of the node correctly allows the port range of 30000 to 32768.

Pod Status Exception and Handling Overview

Last updated：2020-05-25 09:43:06

Users often have to perform complex customization tasks on TKE clusters in order to accommodate their businesses. When Pods do not function properly, it is hard to pinpoint the exact cause. This article aims to provide a starting point for troubleshooting these issues.

Pod Exceptions is a great series of articles that describes how to troubleshoot and solve these issues.

Common Commands

The following is a list of commands commonly used for troubleshooting Pod issues:

- Query Pod status

```
kubectl get pod <pod-name> -o wide
```

- Query Pod YAML configuration

```
kubectl get pod <pod-name> -o yaml
```

- Query Pod events

```
kubectl describe pod <pod-name>
```

- Query container logs

```
kubectl logs <pod-name> [-c <container-name>]
```

Pod Statuses

The following table provides a list of Pod statuses:

Status	Description
`Error`	Error occurred during Pod launch.
`NodeLost`	The node on which the Pod resides is unreachable.

`Unkown`	Pod is unreachable or other unknown exception.
`Waiting`	Pod is waiting to launch.
`Pending`	Pod is waiting to be scheduled.
`ContainerCreating`	Pod containers are being created.
`Terminating`	Pod is being terminated.
`CrashLoopBackOff`	Container exited. Kubelet is restarting it.
`InvalidImageName`	Unable to resolve image name.
`ImageInspectError`	Unable to verify image.
`ErrImageNeverPull`	Policy prohibits image pull.
`ImagePullBackOff`	Trying to pull the image again.
`RegistryUnavailable`	Unable to connect to the image registry.
`ErrImagePull`	General image pull error.
`CreateContainerConfigError`	Unable to create the container configuration used by kubelet.
`CreateContainerError`	Failed to create container.
`RunContainerError`	Failed to launch container.
`PreStartHookError`	preStart hook execution error.
`PostStartHookError`	postStart hook execution error.
`ContainersNotInitialized`	Container not initialized.
`ContainersNotReady`	Container not ready.
`ContainerCreating`	Container is being created.
`PodInitializing`	Pod being initialized.
`DockerDaemonNotReady`	Docker is not ready.
`NetworkPluginNotReady`	Network plugin not ready.

Troubleshooting

Use one of the following articles to troubleshoot your Pod exceptions:

- Pod remains in ContainerCreating or Waiting Status
- Pod Remains in ImagePullBackOff Status
- Pod Remains in Pending Status
- Pod Remains in Terminating Status
- Pod Health Check Fails
- Pod Remains in CrashLoopBackOff Status
- Container Exits

Pod exception troubleshooter
Use Systemtap to Identify Pod Exceptions

Last updated：2022-04-20 19:15:30

This article describes how to use SystemTap to troubleshoot pod issues.

Preparations

Different operating systems have different methods for installing SystemTap and its dependencies. Pick one that suits you.

Ubuntu

1. Run the following command to install SystemTap:

```
apt install -y systemtap
```

2. Run the following command to check for dependencies:

```
stap-prep
```

The following is a sample result:

```
Please install linux-headers-4.4.0-104-generic
You need package linux-image-4.4.0-104-generic-dbgsym but it does not seem to be
available
Ubuntu -dbgsym packages are typically in a separate repository
Follow https://wiki.ubuntu.com/DebuggingProgramCrash to add this repository
apt install -y linux-headers-4.4.0-104-generic
```

3. The above result shows that you need to install dbgsym, which is not in the existing sources. Run the following command to add the third-party source:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C8CAB6595FDFF622

codename=$(lsb_release -c | awk '{print $2}')
```

```
sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
deb http://ddebs.ubuntu.com/ ${codename} main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-security main restricted universe mult
iverse
deb http://ddebs.ubuntu.com/ ${codename}-updates main restricted universe multi
verse
deb http://ddebs.ubuntu.com/ ${codename}-proposed main restricted universe mult
iverse
EOF

sudo apt-get update
```

4. Run the following command after adding the source:

```
stap-prep
```

The following is a sample result:

```
Please install linux-headers-4.4.0-104-generic
Please install linux-image-4.4.0-104-generic-dbgsym
```

5. Run the following command to install the prompted packages:

```
apt install -y linux-image-4.4.0-104-generic-dbgsym
```

```
apt install -y linux-headers-4.4.0-104-generic
```

CentOS

1. Run the following command to install SystemTap:

```
yum install -y systemtap
```

2. For the purpose of this article, we assume that `debuginfo` is not added. Add the following to `/etc/yum.repos.d/CentOS-Debug.repo` and save.

```
[debuginfo]
name=CentOS-$releasever - DebugInfo
baseurl=http://debuginfo.centos.org/$releasever/$basearch/
```

```
gpgcheck=0
enabled=1
protect=1
priority=1
```

3. Run the following command to check for dependencies and install them:

> Note：
> The following command installs `kernel-debuginfo` .

```
stap-prep
```

4. Run the following command to check if the node has multiple versions of `kernel-devel` installed:

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

The returned result is as follows:

```
kernel-devel-3.10.0-327.el7.x86_64
kernel-devel-3.10.0-514.26.2.el7.x86_64
kernel-devel-3.10.0-862.9.1.el7.x86_64
```

If there are multiple versions, keep the one that corresponds to the kernel version. For example, if the current kernel version is `3.10.0-862.9.1.el7.x86_64` , delete all version except `kernel-devel-3.10.0-862.9.1.el7.x86_64` .

> Note：
> - You can use `uname -r` to view the kernel version.
> - Make sure `kernel-debuginfo` and `kernel-devel` are both installed and their versions correspond to the kernel version.

```
rpm -e kernel-devel-3.10.0-327.el7.x86_64 kernel-devel-3.10.0-514.26.2.el7.x86_64
```

Problem Analysis

You can use SystemTap to monitor a process in order to troubleshoot pod issues. This is how it works:

1. SystemTap translates the script into C code and calls gcc to compile the code into the Linux kernel module. It then uses `modprobe` to load the module into the kernel.
2. It uses the script to create kernel hooks and identify the causes of pod issues using the signals captured by the hooks.

Troubleshooting

Step 1: obtain the pids of the containers that restarted automatically in the pod due to exceptions

1. Run the following command to obtain the Container ID:

```
kubectl describe pod <pod name>
```

The returned result is as follows:

```
......
Container ID: docker://5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99c
f1c945
......
Last State: Terminated
Reason: Error
Exit Code: 137
Started: Thu, 05 Sep 2019 19:22:30 +0800
Finished: Thu, 05 Sep 2019 19:33:44 +0800
```

2. Run the following command to query the pid of the main container process using the obtained Container ID:

```
docker inspect -f "{{.State.Pid}}" 5fb8adf9ee62afc6d3f6f3d9590041818750b392dff0
15d7091eaaf99cf1c945
```

The returned result is as follows:

```
7942
```

Step 2: narrow the scope using the container exit code

Use the `Exit Code` in the result of Step 1 to obtain the status code of the last container exit. For the purpose of this article, we will use 137 as an example. The analysis is as follows:

- If the process was killed by an external signal, the exit code should be between 129 and 255.
- An exit code of 137 indicates that the process was killed by `SIGKILL`. However, we still cannot determine the reason why the process exited.

Step 3: use the SystemTap script to identify the reason

Assuming the issue is reproducible, you can use a SystemTap to troubleshoot the problem.

1. Create a file called `sg.stp`. Add the following content and save.

```
global target_pid = 7942
probe signal.send{
if (sig_pid == target_pid) {
printf("%s(%d) send %s to %s(%d)\n", execname(), pid(), sig_name, pid_name, sig
_pid);
printf("parent of sender: %s(%d)\n", pexecname(), ppid())
printf("task_ancestry:%s\n", task_ancestry(pid2task(pid()), 1));
}
}
```

> Note：
> Substitute `pid` with the value of the main container process pid obtained in Step 2. For the purpose of this article, we will use 7942 as an example:

2. Run the following command to execute the script:

```
stap sg.stp
```

When the container process is killed, the script captures the event and outputs the following:

```
pkill(23549) send SIGKILL to server(7942)
parent of sender: bash(23495)
task_ancestry:swapper/0(0m0.000000000s)=>systemd(0m0.080000000s)=>vGhyM0(19491m2.
579563677s)=>sh(33473m38.074571885s)=>bash(33473m38.077072025s)=>bash(33473m38.08
1028267s)=>bash(33475m4.817798337s)=>pkill(33475m5.202486630s)
```

Solution

By observing `task_ancestry`, you can see the parent processes of the stopped process. In the example above, you can see a strange process called `vGhyM0`. This usually indicates that there is a trojan in the system. Take the necessary steps to clean it so your containers can function properly.

Use Exit Code to Identify Pod Exceptions

Last updated：2020-10-16 16:07:23

This document describes how to use exit codes to troubleshoot pod issues.

Querying Pod Exceptions

Run the following command to query pod exceptions:

```
kubectl describe pod <pod name>
```

The returned result is as follows:

```
Containers:
kubedns:
Container ID: docker://5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99c
f1c945
Image: ccr.ccs.tencentyun.com/library/kubedns-amd64:1.14.4
Image ID: docker-pullable://ccr.ccs.tencentyun.com/library/kubedns-amd64@sha256:4
0790881bbe9ef4ae4ff7fe8b892498eecb7fe6dcc22661402f271e03f7de344
Ports: 10053/UDP, 10053/TCP, 10055/TCP
Host Ports: 0/UDP, 0/TCP, 0/TCP
Args:
--domain=cluster.local.
--dns-port=10053
--config-dir=/kube-dns-config
--v=2
State: Running
Started: Tue, 27 Aug 2019 10:58:49 +0800
Last State: Terminated
Reason: Error
Exit Code: 255
Started: Tue, 27 Aug 2019 10:40:42 +0800
Finished: Tue, 27 Aug 2019 10:58:27 +0800
Ready: True
Restart Count: 1
```

`Exit Code` is the status code of the last container exit. If it is not 0, the container exited due to an exception. You can use the exit code to further troubleshoot the problem.

Exit Codes

- A valid exit code is between 0 and 255.
- 0 means the container exited normally.
- If the container exited due to an external signal, the exit code is between 129 and 255. For example, if the operating system sent `kill -9` or `ctrl+c` as the termination signal, the status is `SIGKILL` or `SIGINT`.
- If the container exited due to an internal signal, the exit code is between 1 and 128. However, in some circumstances, the exit code might be between 129 and 255.
- If the specified exit code has a value outside the 0-255 range, such as `exit(-1)`, it is automatically translated to a value in the 0-255 range.

  If the exit code is specified as `code`, it is translated as follows:
  - If the exit code is negative:

    ```
    256 - (|code| % 256)
    ```
  - If the exit code is positive:

    ```
    code % 256
    ```

Typical Exit Codes

- **137**: indicates that the process was killed by `SIGKILL`. Possible reasons are:
  - Pod memory reached `resources.limits`, such as Out of Memory (OOM). Pod resource limits are implemented by using Linux cgroup. If the memory of a pod reaches its limit, cgroup forces it to stop (with a similar effect to `kill -9`). If you use `describe pod`, you can see the value of Reason is `OOMKilled`.
  - If the host does not have sufficient resources (OOM), the kernel stops some processes to free up the memory.

    > **Note：**
    >
    > If the process is stopped due to OOM, cgroup, or the host, you can find relevant records in system logs: Ubuntu system logs are stored in `/var/log/syslog`, whereas CentOS system logs are stored in `/var/log/messages`. You can run the `journalctl -k` command to view system logs in both operating systems.

  - livenessProbe failed, which causes kubelet to stop the pod.
  - Pod stopped by a trojan process.

- **1** and **255**: indicates common issues. Check container logs for further troubleshooting. For example, this could be the result of `exit(1)` or `exit(-1)`. -1 is translated to 255.

Standard Linux Interruption Signals

Linux programs send an exit code when they are interrupted by external signals. The value of the exit code is the value of the interrupt signal plus 128. For example, the value of `SIGKILL` is 9, so the program exit code is 9 + 128 = 137. For more standard interrupt signals, see the following table:

Signal	Status Code Value	Action	Description
`SIGHUP`	1	Term	Hangup detected on controlling terminal or death of controlling process
`SIGINT`	2	Term	Interrupt from keyboard
`SIGQUIT`	3	Core	Quit from keyboard
`SIGILL`	4	Core	Illegal Instruction
`SIGABRT`	6	Core	Abort signal from abort(3)
`SIGFPE`	8	Core	Floating-point exception
`SIGKILL`	9	Term	Kill signal
`SIGSEGV`	11	Core	Invalid memory reference
`SIGPIPE`	13	Term	Broken pipe: write to pipe with no readers; see pipe(7)
`SIGALRM`	14	Term	Timer signal from alarm(2)
`SIGTERM`	15	Term	Termination signal
`SIGUSR1`	30,10,16	Term	User-defined signal 1
`SIGUSR2`	31,12,17	Term	User-defined signal 2
`SIGCHLD`	20,17,18	Ign	Child stopped or terminated
`SIGCONT`	19,18,25	Cont	Continue if stopped
`SIGSTOP`	17,19,23	Stop	Stop process

`SIGTSTP`	18,20,24	Stop	Stop typed at terminal
`SIGTTIN`	21,21,26	Stop	Terminal input for background process
`SIGTTOU`	22,22,27	Stop	Terminal output for background process

C/C++ Exit Codes

`/usr/include/sysexits.h` provides standardized exit codes for C and C++. These codes are described in the following table:

Definition	Status Code	Description
`#define EX_OK`	0	successful termination
`#define EX__BASE`	64	base value for error messages
`#define EX_USAGE`	64	command line usage error
`#define EX_DATAERR`	65	data format error
`#define EX_NOINPUT`	66	cannot open input
`#define EX_NOUSER`	67	addressee unknown
`#define EX_NOHOST`	68	host name unknown
`#define EX_UNAVAILABLE`	69	service unavailable
`#define EX_SOFTWARE`	70	internal software error
`#define EX_OSERR`	71	system error (e.g., can't fork)
`#define EX_OSFILE`	72	critical OS file missing
`#define EX_CANTCREAT`	73	can't create (user) output file
`#define EX_IOERR`	74	input/output error
`#define EX_TEMPFAIL`	75	temp failure; user is invited to retry
`#define EX_PROTOCOL`	76	remote error in protocol
`#define EX_NOPERM`	77	permission denied
`#define EX_CONFIG`	78	configuration error

`#define EX__MAX 78`	78	maximum listed value

Status Code Reference

For the description of more status codes, see the following table:

Status Code	Meaning	Example	Description
1	Catchall for general errors	let "var1 = 1/0"	Miscellaneous errors, such as "divide by zero" and other impermissible operations
2	Misuse of shell builtins (according to Bash documentation)	empty_function() {}	Missing keyword or command
126	Command invoked cannot execute	/dev/null	Permission problem or command is not an executable
127	"command not found"	illegal_command	Possible problem with `$PATH` or a typo
128	Invalid argument to exit	exit 3.14159	**exit** takes only integer args in the range0 - 255 (see first footnote)
128+n	Fatal error signal "n"	*kill -9* `$PPID` of script	**$?** returns137 (128 + 9)
130	Script terminated by Control-C	*Ctl-C*	Control-C is fatal error signal 2, (130 = 128 + 2, see above)
255*	Exit status out of range	exit -1	**exit** takes only integer args in the range0 - 255

Pod Remains in ContainerCreating or Waiting

Last updated：2020-05-25 09:34:49

This article describes the causes that will lead a Pod to become stuck in the `ContainerCreating` or `Waiting` status and how to troubleshoot this issue. Refer to the following instructions to troubleshoot and solve these issues.

Possible Causes

- Incorrect Pod configurations
- Volume failed to mount
- Insufficient disk space
- Node memory fragmentation
- The value of Limit is too small or uses the wrong unit
- Failure to pull the image
- CNI network error
- controller-manager exception
- New Docker installed without completely uninstalling the old version
- Duplicate container names

Troubleshooting

Checking Pod configuration

1. Make sure the image is properly packaged.
2. Make sure the container parameters are configured correctly.

Checking volume mounting

In the following two scenarios, volume mounting issues may cause exceptions:

1. A volume fails to unmount due to Pod float

Analysis

The default volume in a managed Kubernetes cluster is usually a storage class cloud disk. If a node malfunctions and causes kubelet to fail or not be able to communicate with apiserver and the time threshold is reached, the Pods on the node are drained and backup Pods on another node are automatically started. This is called Pod floating. Drained Pods cannot function properly nor are they aware of their states. Therefore, the volume mounted to the node is not

properly unmounted.
cloud-controller-manager requires the volume to unmount properly in order to invoke vendor APIs to unmount disks from the node. Pod floating causes cloud-controller-manager to force unmount a volume after the time threshold is reached and mount it to the node where the Pod is scheduled.

Impact

The Pod may spend an extended period of time in ContainerCreating but will launch successfully.

2. Hitting a subpath bug when mounting configmap/secret

If you modify the content of configmap or secret that is already mounted and the container restarts in place, such as restarting after being killed for failing a liveness check, this bug is triggered.

In this case, the container continuously fails to launch. The error messages are as follows:

```
$ kubectl -n prod get pod -o yaml manage-5bd487cf9d-bqmvm
...
lastState: terminated
containerID: containerd://e6746201faa1dfe7f3251b8c30d59ebf613d99715f3b800740e587e
681d2a903
exitCode: 128
finishedAt: 2019-09-15T00:47:22Z
message: 'failed to create containerd task: OCI runtime create failed: container_
linux.go:345:
starting container process caused "process_linux.go:424: container init
caused \"rootfs_linux.go:58: mounting \\\"/var/lib/kubelet/pods/211d53f4-d08c-11e
9-b0a7-b6655eaf02a6/volume-subpaths/manage-config-volume/manage/0\\\"
to rootfs \\\"/run/containerd/io.containerd.runtime.v1.linux/k8s.io/e6746201faa1d
fe7f3251b8c30d59ebf613d99715f3b800740e587e681d2a903/rootfs\\\"
at \\\"/run/containerd/io.containerd.runtime.v1.linux/k8s.io/e6746201faa1dfe7f325
1b8c30d59ebf613d99715f3b800740e587e681d2a903/rootfs/app/resources/application.pro
perties\\\"
caused \\\"no such file or directory\\\"\"": unknown'
```

For more information on how to resolve this issue, see pr82784.

Checking for insufficient disk space

A Pod uses the CRI APIs to create containers when it launches. This usually involves creating directories and files for the new containers under the data directory. If there is not enough disk space, container creation will fail with the following error messages:

```
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Warning FailedCreatePodSandBox 2m (x4307 over 16h) kubelet, 10.179.80.31 (combine
```

```
d from similar events): Failed create pod sandbox: rpc error: code = Unknown desc
= failed to create a sandbox for pod "apigateway-6dc48bf8b6-l8xrw": Error respons
e from daemon: mkdir /var/lib/docker/aufs/mnt/1f09d6c1c9f24e8daaea5bf33a4230de7db
c758e3b22785e8ee21e3e3d921214-init: no space left on device
```

For more information and further instructions, see Disk Full.

Checking for node memory fragmentation

If node memory is severely fragmented or lacks large page memory, requests for more memory will fail even though there is plenty of memory left. For instructions on troubleshooting and solutions, refer to Memory Fragmentation.

Checking for limit configuration

Error description

- Run `kubectl describe pod` and get the following message:

```
Pod sandbox changed, it will be killed and re-created.
```

- kubelet outputs the following error message:

```
to start sandbox container for pod ... Error response from daemon: OCI runtime
create failed: container_linux.go:348: starting container process caused "proce
ss_linux.go:301: running exec setns process for init caused \"signal: killed\""
: unknown
```

Solution

If the value of limit is too small, Sandbox will fail to run. This will cause the Pod to remain in the ContainerCreating or Waiting status. This is usually a memory limit unit issue.

For example, if you used `m` as the memory limit unit, then Kubernetes reads it as byte. **The correct unit to use is `Mi` or `M`.** If you set a memory limit to 1024m, that translates to 1.024 bytes, which causes a container to be killed by cgroup-oom every it attempts to launch. This results in the Pod remaining in the ContainerCreating state.

Checking for image pull failures

The failure to pull an image produces the same issue. There are many reasons why image pull may fail. The common ones are as follows:

- The wrong image is used
- kubelet cannot access the image registry. For example, images hosted on gcr.io are more difficult to access from Mainland China.
- imagePullSecret is not present or is incorrect when pulling private images.

- Image pull times out due to the size of the image. Adjust the value of `--runtime-request-timeout` and `--image-pull-progress-deadline`.

Pull the image again after checking the above items and check the state of the Pod.

Checking for CNI errors

Make sure that CNI is configured and running properly. If not, you get the following messages:

- Cannot configure Pod network
- Cannot assign Pod IP address

Checking for controller-manager issues

Make sure the Master kube-controller-manager is running properly. Restart it if it is not.

Checking for existing Docker versions

If the node already has Docker installed or installed Docker without completely uninstalling the old Docker, a Pod may encounter the same issue.
For example, if you have installed Docker multiple times using the following command in CentOS:

```
yum install -y docker
```

Due to the incompatibility issue among components of different versions, dockerd continuously fails to create containers. This results in the Pod remaining in the ContainerCreating status. Use `kubectl describe pod` and get the following error messages:

```
Type Reason Age From Message
---- ------ ---- ---- -------
Warning FailedCreatePodSandBox 18m (x3583 over 83m) kubelet, 192.168.4.5 (combine
d from similar events): Failed create pod sandbox: rpc error: code = Unknown desc
= failed to start sandbox container for pod "nginx-7db9fccd9b-2j6dh": Error respo
nse from daemon: ttrpc: client shutting down: read unix @->@/containerd-shim/mob
y/de2bfeefc999af42783115acca62745e6798981dff75f4148fae8c086668f667/shim.sock: rea
d: connection reset by peer: unknown
Normal SandboxChanged 3m12s (x4420 over 83m) kubelet, 192.168.4.5 Pod sandbox cha
nged, it will be killed and re-created.
```

Choose a Docker version to keep and completely uninstall the other versions.

Checking for duplicate container names

Duplicate container names on the same node cause sandbox creation failures, which leads to Pods remaining in the ContainerCreating and Waiting statuses.

Run `kubectl describe pod` and get the following error messages:

```
Warning FailedCreatePodSandBox 2m kubelet, 10.205.8.91 Failed create pod sandbox:
rpc error: code = Unknown desc = failed to create a sandbox for pod "lomp-ext-d8c
8b8c46-4v8tl": operation timeout: context deadline exceeded
Warning FailedCreatePodSandBox 3s (x12 over 2m) kubelet, 10.205.8.91 Failed creat
e pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for po
d "lomp-ext-d8c8b8c46-4v8tl": Error response from daemon: Conflict. The container
name "/k8s_POD_lomp-ext-d8c8b8c46-4v8tl_default_65046a06-f795-11e9-9bb6-b67fb7a70
bad_0" is already in use by container "30aa3f5847e0ce89e9d411e76783ba14accba7eb77
43e605a10a9a862a72c1e2". You have to remove (or rename) that container to be able
to reuse that name.
```

Change container names and make sure there are no duplicate container names on the same node.

Pod Remains in ImagePullBackOff

Last updated：2020-05-25 09:34:50

This article describes the causes that lead to a Pod remaining in the ImagePullBackOff status and how to troubleshoot these issues. Refer to the following instructions for troubleshooting.

Possible Causes

- HTTP registry addresses are not added under insecure-registry
- The self-signed registry CA for HTTPS traffic is not added to the node
- The private image registry failed to authenticate the request
- Damaged image file
- Image pull timed out
- Image not found

Troubleshooting

Checking if HTTP registries are added under insecure-registry

dockerd pulls images from HTTPS registries by default. If you want to use HTTP registries, you need to add them under insecure-registry and restart or reload dockerd to apply the change.

Checking if the self-signed registry CA is added to the node

If your HTTPS registry uses a self-signed CA, dockerd will authenticate the certificate. You can only use the registry if the certificate is successfully authenticated.

To make sure the process succeeds, place the certificate file under:

```
/etc/docker/certs.d/<Registry:port>/ca.crt
```

Checking for private registry configuration issues

If you use a private image registry and the Pod is not configured with an imagePullSecret or uses the wrong imagePullSecret, the registry will refuse the pull requests and the Pod will remain in the ImagePullBackOff status.

Checking for damaged image files

If the image file is damaged before or when it is pushed to the registry, the downloaded image is also damaged. In this case, you need to push the image again.

Checking for image pull timeout

Error description

When multiple Pods are launched at the same time from the same node, all containers pull images and these images are stored in a download queue. If the images in the front of the queue are large in size and take a long time to pull, the images behind them may fail to be pulled due to timeout.

By default, kubelet pulls images one at a time.

```
--serialize-image-pulls Pull images one at a time. We recommend *not* changing th
e default value on nodes that run docker daemon with version < 1.9 or an Aufs sto
rage backend. Issue #10959 has more details. (default true)
```

Solution

If necessary, you can enable concurrent image pulling and set a concurrency limit. The following is an example:

```
--Registry-qps int32 If > 0, limit Registry pull QPS to this value. If 0, unlimit
ed. (default 5)
--Registry-burst int32 Maximum size of a bursty pulls, temporarily allows pulls t
o burst to this number, while still not exceeding Registry-qps. Only used if --Re
gistry-qps > 0 (default 10)
```

Checking if the image exists

If the image does not exist, the Pod may remain in the ImagePullBackOff status. You can identify the issue using kublet logs, as shown by the following:

```
PullImage "imroc/test:v0.2" from image service failed: rpc error: code = Unknown
desc = Error response from daemon: manifest for imroc/test:v0.2 not found
```

Pod Remains in Pending

Last updated：2020-09-18 10:01:38

This article describes the causes that lead to Pods remaining in the Pending status and how to troubleshoot these issues. Refer to the following instructions for troubleshooting.

Error Description

A Pending Pod has not been scheduled to a node. Use `kubectl describe pod <pod-name>` to look up event information, which can be used to analyze the cause.

```
$ kubectl describe pod tikv-0
...
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Warning FailedScheduling 3m (x106 over 33m) default-scheduler 0/4 nodes are avail
able: 1 node(s) had no available volume zone, 2 Insufficient cpu, 3 Insufficient
memory.
```

Possible Reasons

- Insufficient node resources
- nodeSelector and affinity conditions not met
- The node contains a taint that the pod cannot tolerate
- Bugs in earlier versions of kube-scheduler
- kube-scheduler is not running properly
- The stateful application on other usable nodes is not in the same availability zone as the drained node

Troubleshooting

Checking if the node has sufficient resources

Analysis

The following are likely causes of insufficient node resources:

- The CPU utilization is too high.
- There is not enough memory left for allocation.
- There are not enough GPUs left (usually in machine learning and GPU cluster use cases).

Run the following command to query resource allocation information for further analysis:

```
kubectl describe node <node-name>
```

Focus on the following returned items to judge if a node has sufficient resources:

- `Allocatable` : all resources the current node can apply for.
- `Allocated resources` : resources that have been allocated (Allocatable minus all Requests by all Pods on the node).

Impact

The remaining resources a node has is equal to `Allocatable` minus `Allocated resources` . If it is less than the Request from the Pod, then the node does not have enough resources to accommodate the Pod, which means the Scheduler skips the Pod in the Predicates stage. Therefore, the pod is not scheduled to the node.

Checking for nodeSelector and affinity configurations

If the nodeSelector of a Pod specifies a label, the scheduler will only schedule the Pod to a node with that label. If no such node exists, the Pod will not be scheduled. For more information, refer to the official Kubernetes website.

If the Pod has affinity configured and the scheduler cannot find a node that satisfies the affinity conditions, the Pod is not scheduled. Affinity has the following types:

- `nodeAffinity` : affinity to nodes. You can think of this as an enhanced version of nodeSelector. It limits the Pod to the nodes that meet certain conditions.
- `podAffinity` : affinity to Pods. This schedules related Pods to the same node or nodes in the same availability zone.
- `podAntiAffinity` : anti-affinity to pods. This is used to prevent the scheduling of the same type of Pods to the same place in order to avoid single point of failure. For example, you can schedule the Pods that provide DNS service to the cluster to different nodes in order to prevent the DNS service crashes causing business interruptions because a single node fails.

Checking if the node has taints that the Pod cannot tolerate

Analysis

If a node has taints for which the Pod has no corresponding tolerations, the Pod will not be scheduled to that node. You can run `kubectl describe node <node-name>` to query existing node taints, as shown below:

```
$ kubectl describe nodes host1
...
Taints: special=true:NoSchedule
...
```

You can add taints automatically or manually. For more information, refer to Adding Taints.

Solution

This document provides the following solutions. Solution 2 is the most often used.

- Solution 1: delete the taints
  Run the following command to delete the taint named `special` :

  ```
  kubectl taint nodes host1 special-
  ```

- Solution 2: add corresponding tolerations to the Pod

  > **Note：**
  > The following uses a Pod created in the Deployment (named `nginx`) as an example to describe how to add a toleration:

  i. Refer to Logging In to a Linux Instance in Standard Login Mode (Recommended) for instructions on how to log in to the CVM instance that contains `nginx` .
  ii. Run the following command to edit the YAML file:

  ```
  kubectl edit deployment nginx
  ```

  iii. Add tolerations under `spec` in the `template` section. The following adds a toleration for the existing taint `special` :

  ```
  tolerations:
  - key: "special"
  operator: "Equal"
  value: "true"
  effect: "NoSchedule"
  ```

The result should be as follows:

```
template:
  metadata:
    creationTimestamp: null
    labels:
      k8s-app: nginx
      qcloud-app: nginx
  spec:
    containers:
    - image: nginx:latest
      imagePullPolicy: Always
      name: test
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 250m
          memory: 256Mi
      securityContext:
        privileged: false
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    - name: tencenthubkey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
    tolerations:
    - effect: NoSchedule
      key: special
      operator: Equal
      value: "true"
```

iv. Save and exit to complete the process.

Checking if there is a bug in kube-scheduler

There is a bug in earlier versions of `kube-scheduler` that causes Pods to remain in the Pending status. You can solve the issue by upgrading kube-scheduler.

Checking if kube-scheduler is running properly

Check if the Master `kube-scheduler` is running properly. If not, restart the scheduler.

Checking if the stateful application on the drained node is scheduled to a node in another availability zone

If a node fails after a service is deployed, the Pod is evicted and a new Pod is created and scheduled to another node. Pods with mounted disks mounted are usually scheduled to nodes in the same availability zone as the drained node and the disks. However, if the cluster does not have a node that meets the rescheduling requirements, these nodes are not scheduled even if there are nodes in other availability zones that meet the requirements.

The reason that Pods with disks mounted cannot be scheduled to nodes in other availability zones is as follows: Cloud disks can be dynamically mounted to different machines in the same IDC. However, they are not allowed to be mounted to machines in other IDCs to avoid severe I/O degradation due to network latency.

Related Operations

Adding taints

Adding taints manually

Use the following command to add taints manually:

```
$ kubectl taint node host1 special=true:NoSchedule
node "host1" tainted
```

Note：

In some cases, you may not want Pods to be scheduled to a new node before certain configurations are finished. In this case, you can add a taint called `node.kubernetes.io/unschedulable` to the node.

Adding taints automatically

Kubernetes v1.12 Beta provides the feature `TaintNodesByCondition`. With this feature, controller manager will check conditions defined in the node when the node does not run properly. If a condition is met, then the corresponding taint is added automatically.
For example, if the condition of `OutOfDisk` =true is met, then a taint called `node.kubernetes.io/out-of-disk` is added to the node.
Conditions and corresponding taints:

```
Condition Value Taints
-------- ----- ------
OutOfDisk True node.kubernetes.io/out-of-disk
Ready False node.kubernetes.io/not-ready
Ready Unknown node.kubernetes.io/unreachable
MemoryPressure True node.kubernetes.io/memory-pressure
PIDPressure True node.kubernetes.io/pid-pressure
```

```
DiskPressure True node.kubernetes.io/disk-pressure
NetworkUnavailable True node.kubernetes.io/network-unavailable
```

The specific values for each Condition indicate specific meanings as described below:

- If `OutOfDisk` is True, the node is out of storage space.
- If `Ready` is False, the node is unhealthy.
- If `Ready` is Unknown, the node is unreachable. If a node does not report to controller-manager in the time defined by `node-monitor-grace-period` (40s by default), it is marked as Unknown.
- If `MemoryPressure` is True, the node has little available memory.
- If `PIDPressure` is True, the node has too many processes running and it is running out of PIDs.
- If `DiskPressure` is True, the node has little available storage space.
- If `NetworkUnavailable` is `True` , the node cannot communicate with other Pods because the network is not properly configured.

Note：

Taints are added if the above conditions are met. TKE also adds/removes taints actively in the following case:
When a node is created, a taint called `node.cloudprovider.kubernetes.io/uninitialized` is added to it. Then, after successful node initialization, the taint is automatically removed. This is to prevent Pods from being scheduled to an uninitialized node.

Pod Remains in Terminating

Last updated：2022-04-20 19:17:48

This article describes the causes that lead to a Pod remaining in the Terminating status and how to troubleshoot these issues. Refer to the following instructions for troubleshooting.

Possible Causes

- Insufficient disk space
- Files with the **i** attribute exist
- A bug in Docker version 17
- Finalizers exist
- A bug in earlier versions of kubelet list-watch
- Dockerd status and containerd status is not in sync
- A bug in Daemonset Controller

Troubleshooting

Checking if disk space is sufficient

If the disk where the Docker data directory resides is full, Docker will not function properly. It cannot even delete or create containers. Therefore it cannot respond to kubelet's call to delete containers. Use `kubectl describe pod <pod-name>` to query event and get the following messages:

```
Normal Killing 39s (x735 over 15h) kubelet, 10.179.80.31 Killing container with i
d docker://apigateway:Need to kill Pod
```

For solutions and more information, see Disk Full.

Checking to see if files with the i attribute exist

Error description

Use `man chattr` to display a description of the **i** attribute, as shown below:

```
A file with the 'i' attribute cannot be modified: it cannot be deleted or rename
d, no link can be created to this file and no data can be written to the file. On
ly the superuser or a process possessing the CAP_LINUX_IMMUTABLE capability can s
et or clear this attribute.
```

> Note：
> If the container image file itself or files stored in the container have the **i** attribute, they cannot be modified or deleted.

When Pods are deleted, container directories are cleaned. If the directories have files that cannot be deleted, the directories cannot be deleted, which causes the Pods to remain in the Terminating status. In this case, kubelet displays the following error message:

```
Sep 27 14:37:21 VM_0_7_centos kubelet[14109]: E0927 14:37:21.922965 14109 remote_
runtime.go:250] RemoveContainer "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9f
ab281fc09df38257" from runtime service failed: rpc error: code = Unknown desc = f
ailed to remove container "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281f
c09df38257": Error response from daemon: container 19d837c77a3c294052a99ff9347c52
0bc8acb7b8b9a9dc9fab281fc09df38257: driver "overlay2" failed to remove root files
ystem: remove /data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0
391db88534027546868/diff/usr/bin/bash: operation not permitted
Sep 27 14:37:21 VM_0_7_centos kubelet[14109]: E0927 14:37:21.923027 14109 kuberun
time_gc.go:126] Failed to remove container "19d837c77a3c294052a99ff9347c520bc8acb
7b8b9a9dc9fab281fc09df38257": rpc error: code = Unknown desc = failed to remove c
ontainer "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": Erro
r response from daemon: container 19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9
fab281fc09df38257: driver "overlay2" failed to remove root filesystem: remove /da
ta/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db885340275468
68/diff/usr/bin/bash: operation not permitted
```

Solution

- Permanent solution: do not store files with the **i** attribute in container images or set a launched container with the **i** attribute.
- Temporary solution:

1. Use the file path in the kubelet log and run the command `chattr -i <file>` , as shown below:

```
chattr -i /data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f039
1db88534027546868/diff/usr/bin/bash
```

2. Wait for kubelet to restart and try again. You can delete the Pod now.

Checking for the bug in Docker Version 17

Error description

Docker hangs without any response. Running `kubectl describe pod <pod-name>` returns the following results:

```
Warning FailedSync 3m (x408 over 1h) kubelet, 10.179.80.31 error determining stat
us: rpc error: code = DeadlineExceeded desc = context deadline exceeded
```

The cause is likely to be a bug in Docker version 17. You can use `kubectl -n cn-staging delete pod apigateway-6dc48bf8b6-clcwk -force -grace-period=0` to force delete the pod, but you can still see it using `docker ps`.

Solution

Upgrade Docker to version 18. Version 18 uses a new dockerd version and fixed many bugs.

- If the problem persists, submit a ticket for further assistance. **We do not recommend that you force delete the pod** as this may impact your business.

Checking for Finalizers

Error description

If a Kubernetes resource has the `finalizers` metadata, it is created by an application and the `finalizers` field contains an identifier of the application. For example, Rancher-created resources have the `finalizers` identifier.

To delete this type of resource, the application responsible must clean them up and remove the `finalizers` identifiers before they can be deleted.

Solution

Use `kubectl edit` to manually edit the resources to remove `finalizers` before deleting them.

Check for a bug in an earlier version of kubelet list-watch

We discovered that, when you use Kubernetes v1.8.13, kubelet list-watch has a bug that prevents kubelet from receiving event information after deleting a Pod, which means the Pod is not truly deleted. This leads to the Pod remaining in the Terminating status.

Refer to Updating Clusters for instructions on how to update Kubernetes.

Checking if dockerd and containerd are synchronized

Error description

If you use the AUFS storage driver and the disk is full, the kernel may panic and output the following error message:

```
aufs au_opts_verify:1597:dockerd[5347]: dirperm1 breaks the protection by the per
mission bits on the lower branch
```

If this happens, it may lead to status synchronization issues, and dockerd logs may contain records similar to the following:

```
Sep 18 10:19:49 VM-1-33-ubuntu dockerd[4822]: time="2019-09-18T10:19:49.903943652
+08:00" level=error msg="Failed to log msg \"\" for logger json-file: write /opt/
docker/containers/54922ec8b1863bcc504f6dac41e40139047f7a84ff09175d2800100aaccbad1
f/54922ec8b1863bcc504f6dac41e40139047f7a84ff09175d2800100aaccbad1f-json.log: no s
pace left on device"
```

Analysis

You can use one of the following methods to find out if dockerd and containerd are in sync.

- Use `describe pod` to obtain the ID of the container. Then, use `docker ps` to query the status of the container and see if it matches the status from dockerd.
- Use `docker-container-ctr` to query the container status in containerd, as shown below:

  ```
  $ docker-container-ctr --namespace moby --address /var/run/docker/containerd/do
  cker-containerd.sock task ls |grep a9a1785b81343c3ad2093ad973f4f8e52dbf54823b8b
  b089886c8356d4036fe0
  a9a1785b81343c3ad2093ad973f4f8e52dbf54823b8bb089886c8356d4036fe0 30639 STOPPED
  ```

If the status of the container in containerd is `stopped` or empty and it is `running` in dockerd, then the container status is not synchronized between dockerd and containerd.

Solution

- Temporary solution: run `docker container prune` or restart dockerd.
- Permanent solution: use containerd instead of both containerd and dockerd to work around the bug in dockerd.

Checking for the Daemonset Controller bug

Kubernetes 1.10 and 1.11 have a bug that causes Daemonset Pod to remain in the Terminating status. In this case, Daemonset Controller reuses the predicates logic of scheduler which sorts the nodeSelector array (passed as pointer parameters) from nodeAffinity. This results in spec being different from that stored by apiserver. At the same time, Daemonset Controller uses spec to calculate the hash of Daemonset for version control purposes.
This difference in parameter values causes the Pod to get stuck in a loop of launching and stopping.

Solution

- Temporary solution: make sure rollingUpdate Daemonset uses nodeSelector rather than nodeAffinity.
- Permanent solution: refer to Updating Clusters for instructions on how to update Kubernetes to 1.12.

Pod Health Check Fails

Last updated：2020-05-25 09:34:50

This article describes the causes of health check failures and how to troubleshoot them. Refer to the following instructions to troubleshoot and solve these issues.

Error Description

Kubernetes health checks include readiness checks (readinessProbe) and liveness checks (livenessProbe). Different health check failures have different symptoms:

- Pod IP addresses are removed from Service and traffic is not directed to the Pods that failed readiness check.
- kubelet stops a Pod and tries to restart it.

There are many reasons why health checks may fail. For example, the application may have a bug that prevents it from responding to health checks. If a Pod becomes `Unhealthy`, following these instructions to troubleshoot it:

Possible Causes

- Improper health check configuration
- Node overload
- Container process stopped by a trojan
- The listening port of a container internal process fails
- SYN backlog setting too low

Troubleshooting

Checking your health check configuration

An improper health check configuration may cause health checks to fail. For example, if `initialDelaySeconds` (the period of time to wait before probing a container for the first time after the container starts) is too low and a container is slow to start, this will cause the probe to start before the container finishes startup. If, at the same time `successThreshold` is set to 1, then the health check is performed once and stopped. As a result, the Pod is stuck in a loop where it is repeatedly stopped and restarted.

Checking if the node is overloaded

High CPU usage (such as 100%) causes the process to be unable to send or receive packets, which leads to timeout and health check failures. See High Workload for more information on how to troubleshoot this issue.

Checking if the container process was stopped by a trojan

See Using Systemtap to Troubleshoot Pod Exceptions for more information on how to troubleshoot this issue.

Checking if the listening port of the container internal process stopped working

Use `netstat -tunlp` to check if the port is still listening. From the results we can conclude: if the port stops listening, health check probe requests are reset, as shown by the following:

```
20:15:17.890996 IP 172.16.2.1.38074 > 172.16.2.23.8888: Flags [S], seq 96880261,
win 14600, options [mss 1424,nop,nop,sackOK,nop,wscale 7], length 0
20:15:17.891021 IP 172.16.2.23.8888 > 172.16.2.1.38074: Flags [R.], seq 0, ack 96
880262, win 0, length 0
20:15:17.906744 IP 10.0.0.16.54132 > 172.16.2.23.8888: Flags [S], seq 1207014342,
win 14600, options [mss 1424,nop,nop,sackOK,nop,wscale 7], length 0
20:15:17.906766 IP 172.16.2.23.8888 > 10.0.0.16.54132: Flags [R.], seq 0, ack 120
7014343, win 0, length 0
```

As shown above, health check probe request exceptions lead to health check failure. Possible causes are:
If a node has multiple Pods that use `hostNetwork` to listen on the same host port, only one Pod will be able to listen while the other Pods will fail to listen but do not exit. That means they will all be probed by health checks and all Pods but one will fail the health check.

Checking if SYN backlog value is too low

Error description

The value of SYN backlog is the size of the SYN queue. If this value is set too low and many new connection requests are received in a short time, the majority of the requests will fail. You can use `netstat -s | grep TCPBacklogDrop` to get the number of failed requests.

Solution

Once you are sure the requests failed due to the value of SYN backlog, increase the value. The kernel parameter to use is `net.ipv4.tcp_max_syn_backlog` .

Pod Remains in CrashLoopBackOff

Last updated：2022-04-18 16:18:24

This article describes the reasons that may cause a Pod to fail and enter the CrashLoopBackOff status and how to troubleshoot the issues. Refer to the following instructions to troubleshoot and solve these issues.

Error Description

If a Pod's status is `CrashLoopBackOff` , this means the Pod was launched but exited with exceptions. When this happens, unless the Pod's restartPolicy is `Never` , the Pod will be restarted and the `RestartCounts` of the Pod will usually be greater than 0. In this case, first see Using Exit Code to Troubleshoot Pod Exiting with Exceptions for information on using the exit code to narrow down the range of possible problems.

Possible Causes

- Container process exited
- System OOM
- cgroup OOM
- Node memory fragmentation
- Health check failed

Troubleshooting

Making sure the containers are not killed

When a container exits, the exit code usually is between 0 and 128. The cause of the exception may be a bug or other reason.
Refer to Container Exits for more information on how to further troubleshoot such problems.

Checking for system OOM

Analysis

If system OOM occurs, the exit code of the containers will be 137, indicating they exited due to the `SIGKILL` signal. The kernel will display the following error message:

```
Out of memory: Kill process ...
```

This can occur when other non-Kubernetes processes deployed on the node use too much memory, or not enough memory was assigned to kubelet using `--kube-reserved` and `--system-reserved`, leaving too little headroom for other non-container processes.

> Note：
> The total memory usage of all Pods on a node will not exceed the value of `cgroup` defined in `/sys/fs/cgroup/memory/kubepods` (`cgroup = capacity - "kube-reserved" - "system-reserved"`). In most cases, if memory is properly divided and the non-container processes (such as kubelet, dockerd, kube-proxy and sshd) on the same node do not use up the reserved memory, system OOM should not occur.

Solution

Adjust memory allocation according to your needs to avoid this issue.

Checking for cgroup OOM

Error description

If the Pod exited due to cgroup OOM, the value of `Reason` under Pod events will be `OOMKilled`, indicating the actual usage of the container memory exceeded the limit. The kernel log will show the `Memory cgroup out of memory` error message.

Solution

Adjust limit according to your needs.

Node memory fragmentation

If node memory is severely fragmented or lacks large page memory, requests for more memory will fail even though there is plenty of memory left. For instructions on troubleshooting and solutions, refer to Memory Fragmentation.

Health check failures

For information on how to troubleshoot this issue, see Health Check Failures.

Pod Kept Restarting with Traffic Exception

Last updated：2022-12-08 17:25:19

Problems

The Pod suddenly kept restarting, with abnormal traffic coming in.

Cause

1. The Pod drifted to another node for start, as its previous node was abnormal.
2. After recreation, the Pod started slowly due to a faulty dependent service of the basic image. As both `ReadinessProbe` and `LivenessProbe` were configured, it was highly likely that all health checks failed more times than the upper limit set in `LivenessProbe`, thereby leading to a restart.
3. The Pod was configured with `preStop` for graceful termination, indicating to run `preStop` before restart. The graceful termination took a long time, and during `preStop` execution, `ReadinessProbe` kept probing.
4. TCP probe was used. During graceful termination, the TCP probe succeeded (port listening existed before the process was completely closed), but the process no longer processed new requests.
5. The result of `ReadinessProbe` but not `LivenessProbe` determines whether a Pod is ready. As `ReadinessProbe` was successful during `preStop` execution, the Pod became ready.
6. The Pod was ready but couldn't process requests, leading to business exceptions.

Summary

1. The Pod kept restarting due to a slow start and liveness probe. You need to prolong `initialDelaySeconds` or StartProbe to protect containers that start slowly.
2. The TCP probe method cannot reflect the actual health status of the business. During graceful termination, `ReadinessProbe` succeeds and lets traffic in, which will not be handled by the business, leading to traffic exceptions. We recommend you use a better probe method, where the business provides the HTTP liveness probe to check the actual health status of the business.

Container Process Exits

Last updated：2020-05-25 09:43:07

This article describes several scenarios that can cause containers to exit and provides instructions on how to troubleshoot these issues.

Error Description

When a container exits (not killed by external sources), the exit code is usually between 0 and 128. 0 indicates a normal exit and 1-127 indicates exits due to exceptions. For example, if an application detects that its launch parameters or conditions are not met or the application panics but the exception is not handled, the application will exit.
Refer to Using Exit Codes to Troubleshoot Pod Exceptions for more information on container exit code details.

Possible Causes

- Failure to resolve DNS
- Application configuration issues

Troubleshooting

Checking for DNS resolution failures

If the application relies on the cluster DNS service, unresolved DNS requests will cause the application to throw exceptions and exit. For example, if the application needs to connect to the database when it launches and the database uses a service name or external domain name that needs to be resolved by a DNS server. Unresolved DNS requests lead to application exception and exit. Possible causes are as follows:

- The cluster network is not functioning properly, and Pods cannot connect to the DNS service.
- The DNS service not functioning properly and cannot respond to requests.
- The service name or domain name is unresolvable.

Checking for application configuration issues

If the application is not configured properly, this can also result in the application exiting. Possible causes are as follows:

- The configuration file is not correctly formatted. The application fails to resolve the configuration when launching, which leads to exceptions and exit.
- The configuration values do not meet requirements. For example, a missing required field value will cause the configuration to fail verification, which leads to application exceptions and exit.

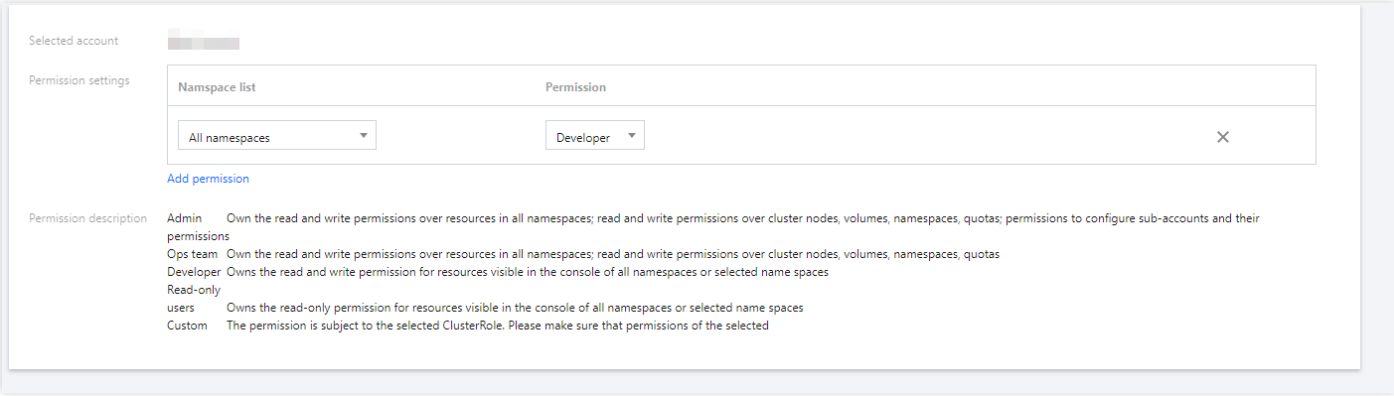
Authorizing Tencent Cloud OPS Team for Troubleshooting

Last updated：2023-05-22 10:15:09

Tencent Cloud OPS team is not allowed to log in to your cluster for troubleshooting without your permission. If you need Tencent Cloud OPS team to assist in troubleshooting, please refer to the following steps to grant Tencent Cloud OPS team related permissions. You can cancel the permissions authorized to Tencent Cloud OPS team at any time.

Grant permissions to Tencent Cloud though console

1. Log in to the TKE console.
2. On **Cluster Management** page, select the cluster where Tencent Cloud assistance is needed.
3. On the cluster details page, select **Authorization Management** > **Authorize Tencent Cloud OPS team**.
4. When configuring the cluster RBAC, select the operation permissions that you want to authorize to Tencent Cloud OPS team.



5. After the configuration is completed, you can check the progress in My Tickets.

> Note：
> Tencent Cloud OPS team is not allowed to log in to your cluster for troubleshooting without your permission. If you need Tencent Cloud OPS team to assist in troubleshooting, you can grant Tencent Cloud OPS team related permissions. You can also cancel the permissions authorized to Tencent Cloud OPS team at any time.
> You can withdraw permissions authorized to Tencent Cloud OPS team by deleting relevant resources (ClusterRoleBinding/tkeopsaccount-ClusterRole, ServiceAccount/tkeopsaccount, and Sercet/tkeopsaccount-token-xxxx).

Grant permissions to Tencent Cloud OPS team through Kubernetes API

You can grant permissions to Tencent Cloud OPS team by creating the following Kubernetes resources.

ServiceAccount: authorize Tencent Cloud OPS team to access cluster credential

```
kind: ServiceAccount
apiVersion: v1
metadata:
name: tkeopsaccount
namespace: kube-system
labels:
cloud.tencent.com/tke-ops-account: tkeops
```

ClusterRoleBinding/RoleBing: rules on granting Tencent Cloud OPS team permissions

Note：

1. Name and label should be created according to the following rule.
2. roleRef can be replaced with the permissions you want to grant to Tencent Cloud OPS team.

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
annotations:
cloud.tencent.com/tke-ops-account: tkeops
labels:
cloud.tencent.com/tke-ops-account: tkeops
name: tkeopsaccount-ClusterRole
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: tke:admin
subjects:
- kind: ServiceAccount
name: tkeopsaccount
namespace: kube-system
```

(Optional) ClusterRole/Role: permissions authorized to Tencent Cloud OPS team

If there is relevant ClusterRole/Role in the cluster, you can use ClusterRoleBinding/RoleBinding to associate. Policies will be created automatically if you authorize through console.

- Admin permissions
- Read-only

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
labels:
cloud.tencent.com/tke-rbac-generated: "true"
name: tke:admin
rules:
- apiGroups:
- '*'
resources:
- '*'
verbs:
- '*'
- nonResourceURLs:
- '*'
verbs:
- '*'
```

Engel Ingres appears in Connechtin Reverside

Last updated：2022-09-26 16:54:29

Symptom

When you are using NGINX Ingress and reducing the number of NGINX Ingress Controller replicas, the problem of "Connection Refused" may occur. In this case, RSs are unbound from CLB instances in batches, and TCP/UDP listeners stop forwarding existing connections.

Possible Causes

View the source code of NGINX Ingress and you can see that the workloads of NGINX Ingress Controller have no graceful shutdown capabilities. Therefore, a Pod directly exits after receiving the kill signal.

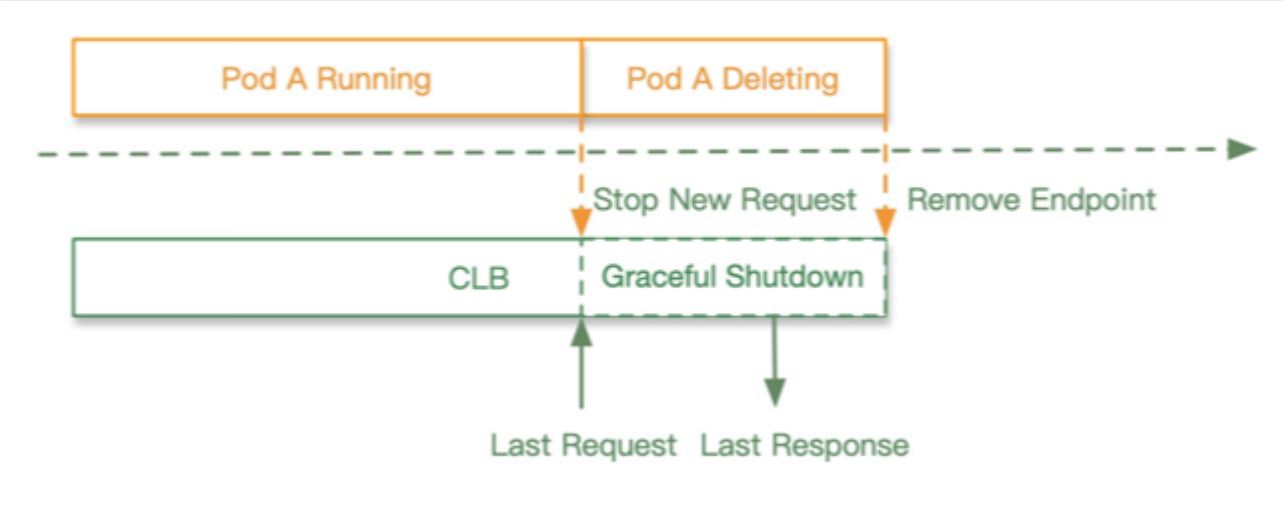
```
func main() {
	err := exec.Command("bash", "-c", "pkill -SIGTERM -f nginx-ingress-controller" .Run()
	if err != nil {
		klog.ErrorS(err, "terminating ingress controller")
		os.Exit(1)
	}

	// wait for the NGINX process to terminate
	timer := time.NewTicker(time.Second * 1)
	for range timer.C {
		if !nginx.IsRunning() {
			timer.Stop()
			break
		}
	}
}
```

Solutions

If you use TKE's graceful service shutdown capabilities, when a Pod needs to be deleted, it can process the received requests, and inbound traffic is turned off while outbound traffic is still on. Outbound traffic will not be turned off until all

existing requests are processed and the Pod is deleted. The Pod is deleted after the graceful shutdown period ends.



Troubleshooting

> Note：
> This is only effective in the direct access mode. Check whether your cluster supports direct access.

Step 1

Use an annotation to indicate the use of graceful shutdown in the `****-ingress-nginx-controller` Service in the `kube-system` namespace.

The following is an example of using an annotation to indicate the use of graceful shutdown. For more information on Service annotations, see Service Annotation.

```
kind: Service
apiVersion: v1
metadata:
annotations:
service.cloud.tencent.com/direct-access: "true" ## Enable CLB-to-Pod direct acces
s
service.cloud.tencent.com/enable-grace-shutdown: "true" # Indicate the use of gra
ceful shutdown
name: my-service
spec:
selector:
app: MyApp
```

Step 2

Add a sleep period before the `wait-shutdown` of the `****-ingress-nginx-controller` Deployment in the `kube-system` namespace.

```
lifecycle:
preStop:
exec:
command:
- sleep # Add a sleep period
- 30s # Add a sleep period
- /wait-shutdown # Add a sleep period before this line
```

For more information, see Graceful Service Shutdown. If the problem persists, submit a ticket for assistance.

CLB Loopback

Last updated：2022-04-06 10:29:27

Background

In order to shorten the linkage and improve the performance when the business in a cluster accesses a Service of the LoadBalancer type through Tencent Cloud CLB, the community edition of Kube-Proxy binds the CLB IP to the local dummy ENI in IPVS mode, so that the traffic short-circuits on the node instead of going through CLB and thus is directly forwarded to the endpoint locally. However, this optimization conflicts with the health check mechanism of CLB. The following makes a detailed analysis and describes corresponding solutions.

Similarly, when you use TKE, CLB loopback may occur, causing service inaccessibility or several seconds of latency during access to an Ingress. This document describes the symptoms, causes, and solutions of this problem.

Issue Description

CLB loopback may cause the following symptoms:

- No matter whether you are in iptables or IPVS mode, when you access an Ingress on the cluster private network, a 4-second latency or inaccessibility occurs.
- In IPVS mode, when you access a LoadBalancer service on the cluster private network within your cluster, it is completely inaccessible, or the connection is unstable.

Workaround

Avoiding layer-4 loopback

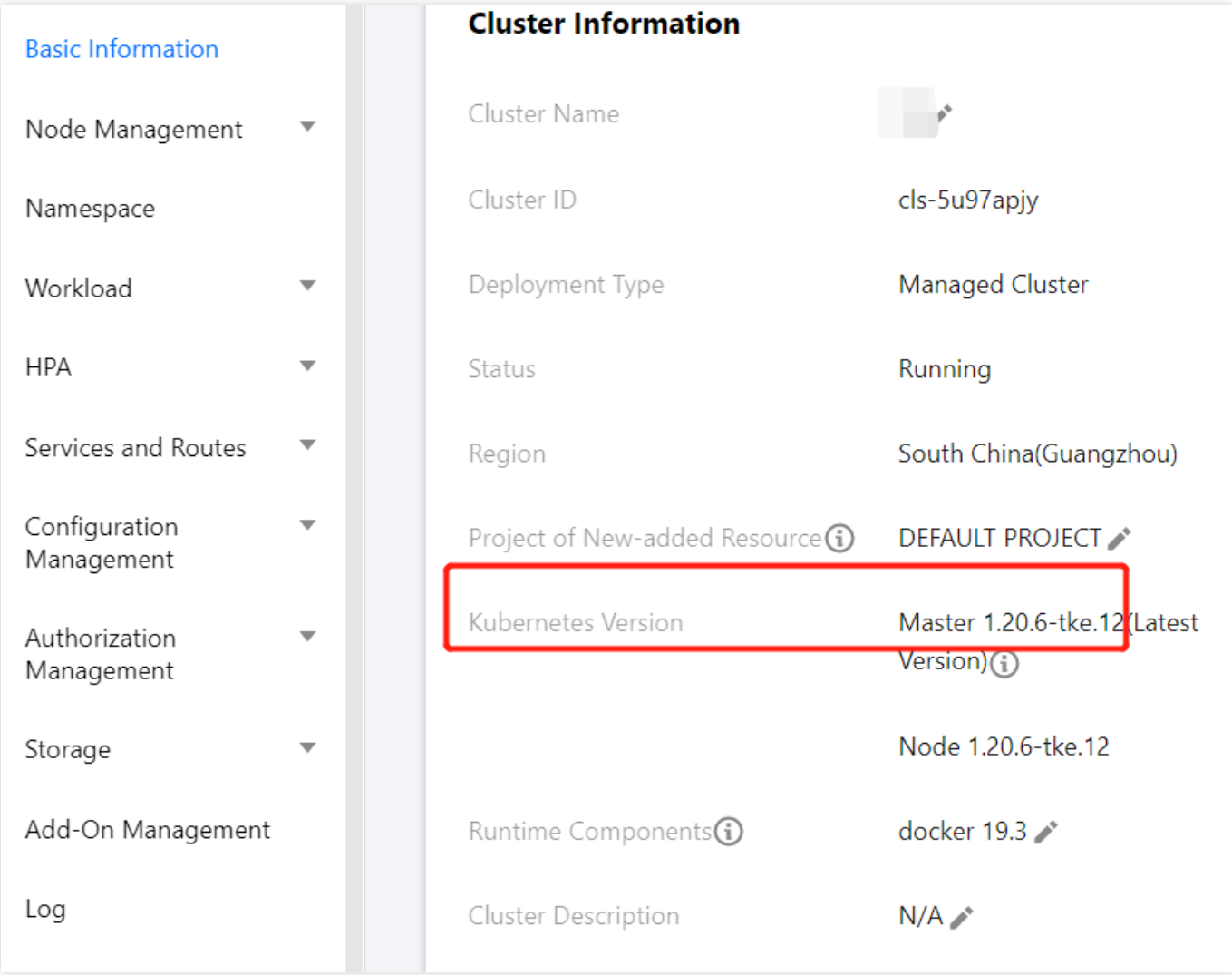
> Note：
> Avoiding the layer-4 loopback problem requires CLB to support non-VIP health check source IP. This feature is currently in beta test. To try it our, submit a ticket for assistance.

To solve the loopback problem that may be encountered when using the Service, follow the steps below:

1. Check whether kube-proxy is on the latest version, and if not, upgrade it as follows:

i. To solve this problem, kube-proxy needs to support binding the LB address to the IPVS ENI. You can find version numbers that support this capability in TKE Kubernetes Revision Version History, such as v1.20.6-tke.12, v1.18.4-tke.20, v1.16.3-tke.25, v1.14.3-tke.24, and v1.12.4-tke.30.

ii. Determine the version of the cluster: you can view the version number of the current cluster on the **Basic Information** page in the cluster as shown below:



iii. Find the DaemonSet named kube-proxy under the kube-system namespace, update the version number of its image, and select a version that supports this capability or a later version. For example, for a cluster on v1.18, you need to select an image version later than v1.18.4-tke.20.

2. Annotate all Services:

```
service.cloud.tencent.com/prevent-loopback: "true"
```

Effect:

- kube-proxy will bind the CLB VIP to the local system based on this information to solve the loopback problem.

- service-controller will call the CLB API and change its health detection IP to an IP in the 100.64 IP range to solve the health check problem.

Avoiding layer-7 loopback

To solve the loopback problem that may be encountered when using Ingress, follow the steps below:

1. Submit a ticket to apply for the CLB layer-7 SNAT and keepalive capabilities.

> Note：
> These capabilities will enable persistent connection. Then, persistent connections will be used between CLB and real server, and CLB will no longer pass through the source IP, which can be obtained from XFF. To ensure normal forwarding, enable the "Allow by default" feature in the CLB security group or allow `100.127.0.0/16` in the CVM security group. For more information, see Configuring HTTP Listener.

2. Use the capabilities of TkeServiceConfig to enhance the configuration capabilities of the Ingress. For existing Ingresses, you need to add an annotation: `ingress.cloud.tencent.com/tke-service-config-auto: "true"` . For more information, see Ingress Annotation.
   Purpose: this automatically generates the corresponding TkeServiceConfig for the Ingress and provides additional configuration for the Ingress.
   Effect: the Ingress will generate a resource named `<ingressname>-auto-ingress-config` of the TkeServiceConfig type.

3. Add a field in the generated TkeServiceConfig named `<ingressname>-auto-ingress-config` . The field name is `spec.loadBalancer.l7Listeners[].keepaliveEnable` , and the field value is 1. Note that this field needs to be added for each port as shown below:

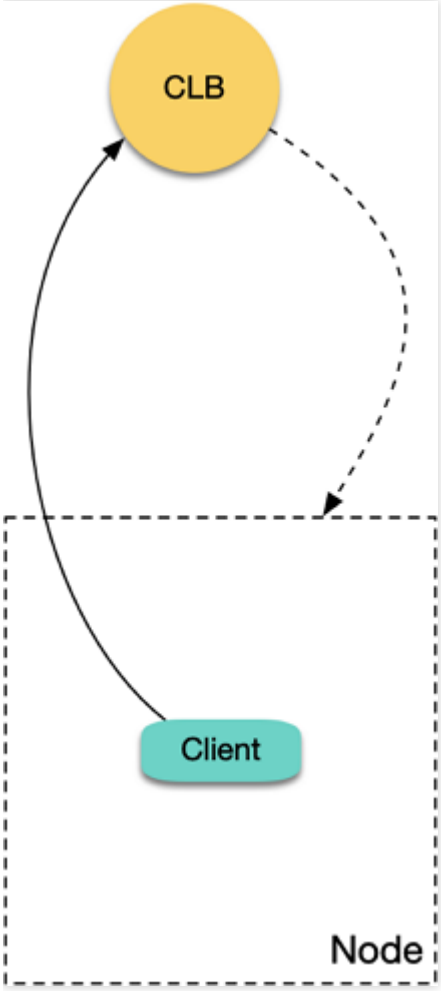
```
apiVersion: cloud.tencent.com/v1alpha1
kind: TkeServiceConfig
metadata:
  name: jetty-ingress-config
  namespace: default
spec:
  loadBalancer:
    l7Listeners:
    - protocol: HTTP
      keepaliveEnable: 1
      port: 80
      domains:
      - domain: ""      # domain
        rules:
        - url: "/health"
          forwardType: HTTP #  HTTP
          healthCheck:
            enable: false
    - protocol: HTTPS
      keepaliveEnable: 1
      port: 443
      domains:
      - domain: "sample.tencent.com"
        rules:
        - url: "/"
          forwardType: HTTPS #  HTTPS
          session:
            enable: true
            sessionExpireTime: 3600
          healthCheck:
```

For more information, see Using TKEServiceConfig to Configure CLBs.

Causes

The root cause is that when CLB forwards a request to the real server, the packet source and target IPs are both on the same node, but Linux will ignore the received packets whose source IP is the local IP by default, causing the data

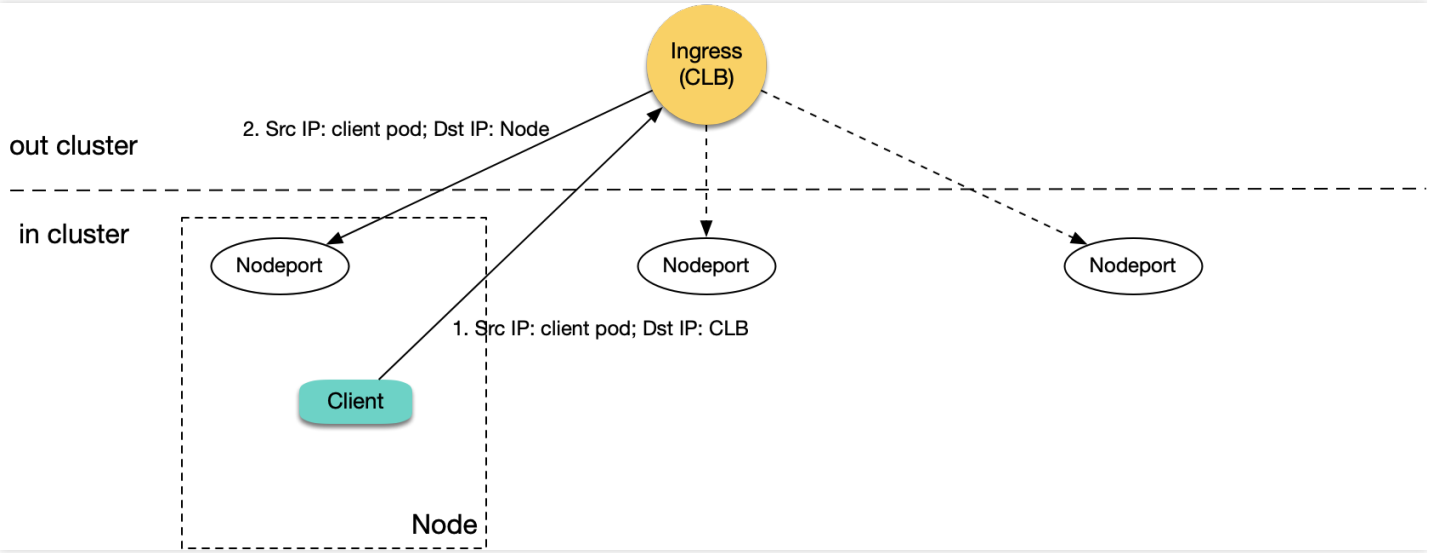
packet to be looped within the CVM instance as shown below:



Analysis of layer-7 loopback problem

If you use a TKE CLB Ingress, a CLB instance and layer-7 listener rules (HTTP/HTTPS) for ports 80 and 443 will be created for each Ingress resource, and the same NodePort of each corresponding TKE node will be bound to each Ingress location as the real server (a location corresponds to a Service, and each Service uses the same NodePort of each node to expose the traffic). CLB forwards the request to the corresponding real server (i.e., NodePort) according to the location matched by the request, and the traffic will be forwarded to the corresponding backend Pod after passing the NodePort and the Kubernetes iptables or IPVS. When a Pod in the cluster accesses the private network

Ingress in the cluster, CLB will forward the request to the corresponding NodePort of a node in the cluster.



As shown above, when the node whose traffic is forwarded is the node of the client that sends the request:

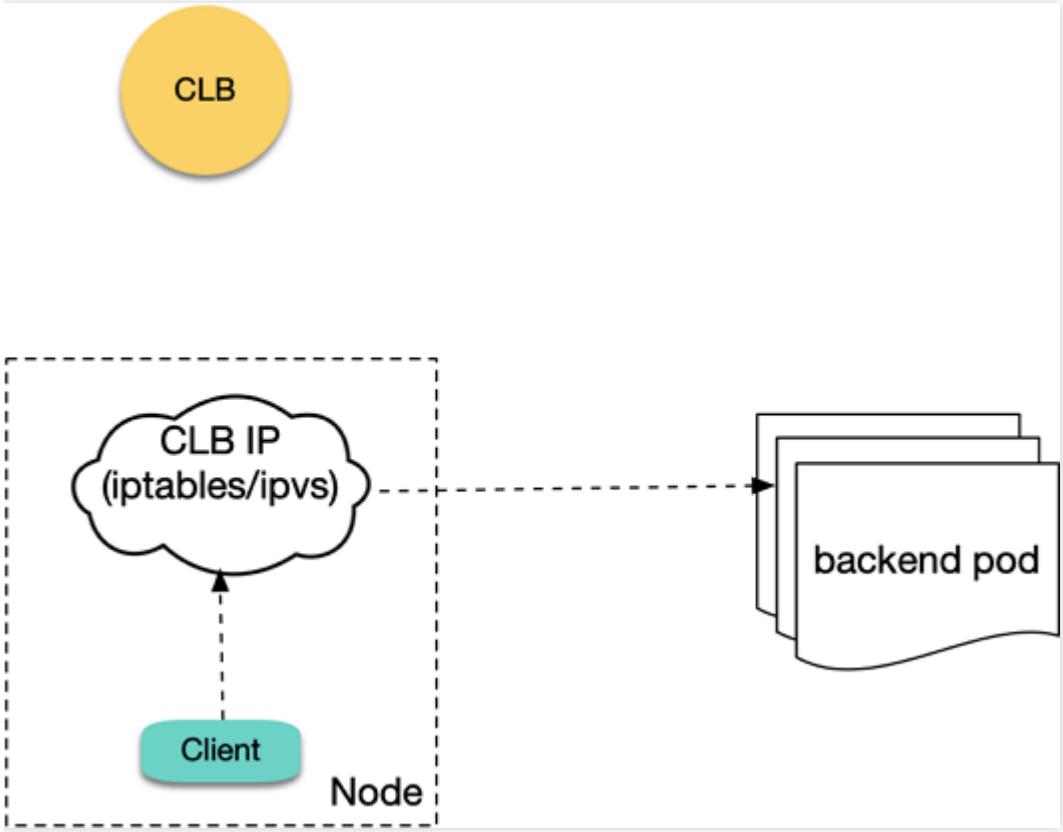
1. A Pod in the cluster accesses CLB, and CLB forwards the request to the corresponding NodePort of any node.
2. When the packet reaches the NodePort, the target IP is the node IP, and the source IP is the actual IP of the client Pod. As CLB does not perform SNAT, it will pass through the actual source IP.
3. As the source and target IPs are both on the same server, loopback will occur, and CLB cannot receive response from the real server.

The most frequent fault of access to an Ingress in the cluster is a latency of several seconds. It is because if a layer-7 CLB instance's request to a real server times out (in about 4 seconds), the instance will try the next real server. Therefore, if you set a long timeout period on the client, loopback may occur with a symptom of slow request response with a several-second latency. If your cluster has only one node, CLB has no real server for retry, and the symptom will be inaccessibility.

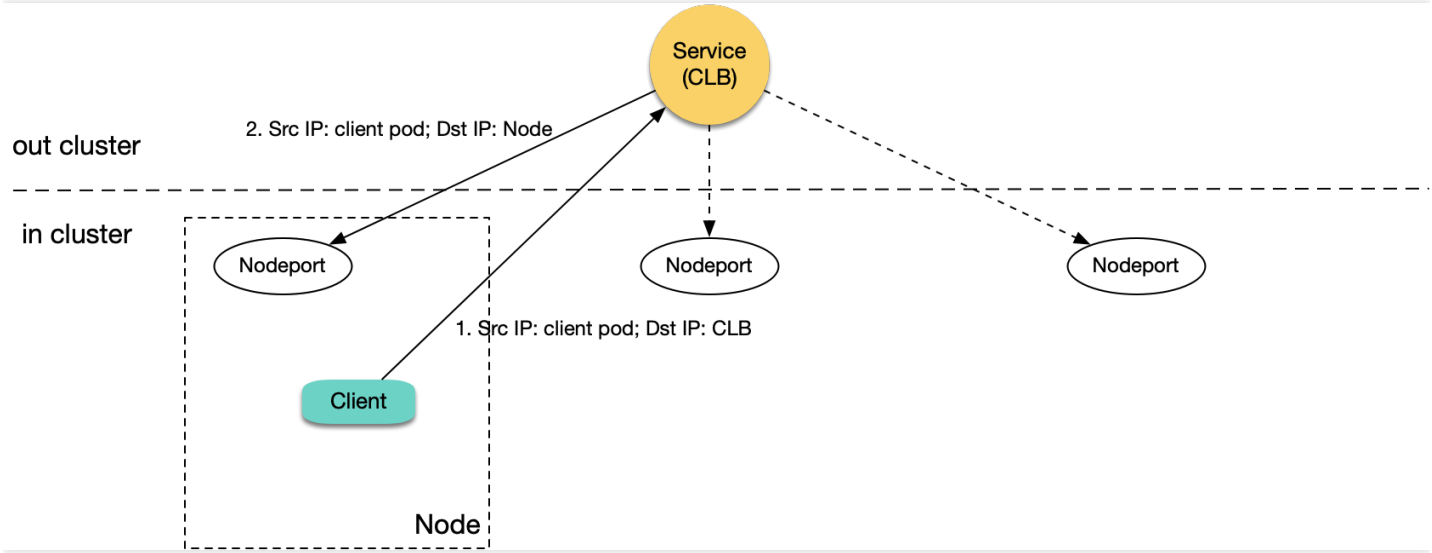
Analysis of layer-4 loopback problem

If you use a LoadBalancer-type private network Service to expose your service, a private network CLB instance and the corresponding layer-4 listener (TCP/UDP) will be created. If a Pod in your cluster accesses the `EXTERNAL-IP` (i.e., CLB IP) of the LoadBalancer-type Service, the native Kubernetes will not actually access the LB but will directly

forward the traffic to the backend Pod through iptables or IPVS without passing CLB as shown below:



Therefore, the native Kubernetes logic has no loopback errors. However, in IPVS mode of TKE, the access request packet of a client to a CLB IP will be actually sent to CLB. Loopback will occur when a Pod accesses the CLB IP of a LoadBalancer-type Service in the same cluster in IPVS mode, which is similar to the aforementioned private network Ingress loopback error as shown below:



The differences lie in that when loopback occurs, the layer-4 CLB instance will not try the next real server, and the symptom is usually unstable connection. If the cluster has only one node, complete inaccessibility will be caused.

Why doesn't the IPVS mode of TKE use the similar forwarding logic of the native Kubernetes (i.e., the traffic is directly forwarded to the backend Pod without passing the LB)?

In the legacy TKE IPVS mode, the cluster uses a private network LoadBalancer Service to expose your service, and the health checks performed by the private network CLB on the backend NodePort will all fail. Below are the causes:

- IPVS mainly runs on the INPUT chain, so the forwarded VIPs (Service cluster IP and `EXTERNAL-IP`) need to be used as the local server IP to make the packet enter the INPUT chain and get processed by IPVS.
- kube-proxy binds both the cluster IP and `EXTERNAL-IP` to a dummy ENI named `kube-ipvs0` , which is only used to bind the VIPs (the kernel automatically generates the local route for it) instead of receiving traffic.
- The source IP of the health check packets in private network CLB sent to the NodePort is the CLB VIP, and the target IP is the node IP. When a health check packet arrives at the node, the node will discover that the source IP is the local server IP (as it is bound to `kube-ipvs0`) and discard it. Therefore, CLB health check packets can never get the response, that is, all checks will fail. Although CLB has the all-dead-all-alive logic (failure of all checks are deemed as that all requests can be forwarded), this problem still equals that the check is of no use and will cause some exceptions in certain cases.

To solve the aforementioned problems, the solution of TKE is not to bind `EXTERNAL-IP` to `kube-ipvs0` in IPVS mode; that is, packets of a cluster Pod's access requests to the CLB IP will not enter the INPUT chain; instead, they will be directly sent by the node ENI to CLB actually. In this way, the health check packets will not be discarded as packets with the local server IP when entering the node, and the health check response packets will not enter in the INPUT chain and then get looped within it.

Although this method fixes the problem of CLB health check failure, it also makes cluster Pods' access request packets to CLB actually arrive at CLB. As the service in the cluster is accessed, the packets will be forwarded back to a cluster node, which also may cause loopback.

> Note：
> The problem has been submitted to the community, but there is still no solution.

FAQs

Why public network CLB does not have loopback?

There is no loopback issue if you use a public network Ingress or public network LoadBalancer Service. This is because the source IP of the packets received by public network CLB is the public network egress IP of a CVM

instance, but the CVM instance cannot sense its own public network IP internally. When a packet is forwarded to the CVM instance, the public network source IP will not be considered as the local server IP, so there will not be loopback.

Does CLB have any mechanism to avoid loopback?

Yes. CLB will determine the source IP and will not consider forwarding the request to the real server with the same IP; instead, it will forward the request to another real server. However, the source Pod IP is different from the real server IP, and CLB does not know whether the two IPs are on the same node, so the request may still be forwarded and cause loopback.

Can anti-affinity deployment of the client and server avoid loopback?

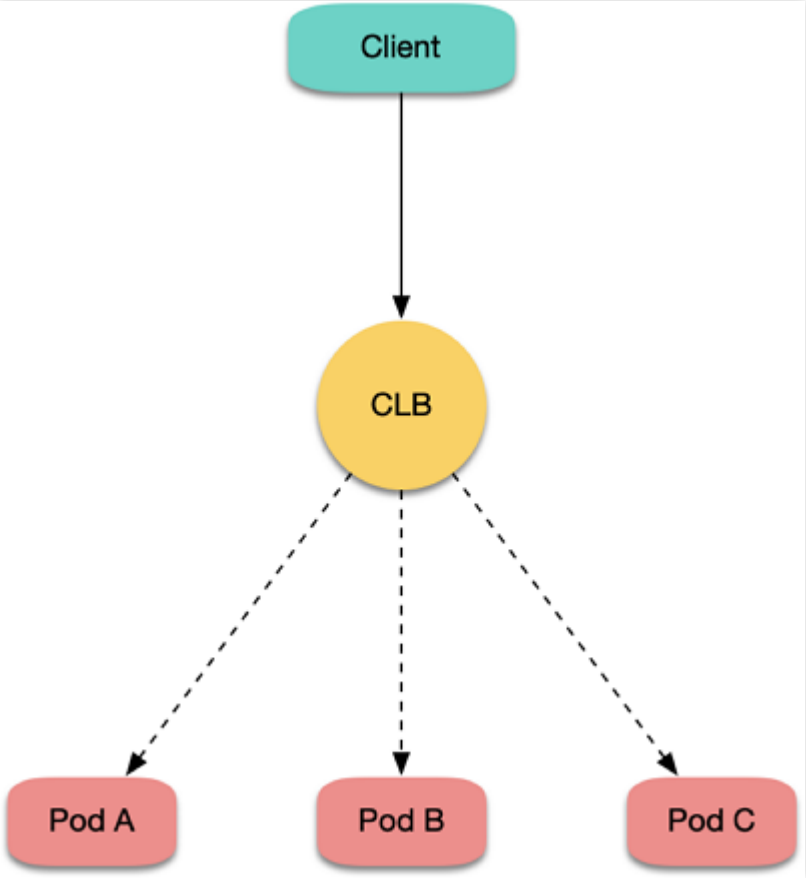
If you deploy a client and a server through anti-affinity so that they are not on the same node, can CLB loopback be avoided?

By default, LB is bound to a real server through a node NodePort, and a request may be forwarded to NodePort of any node. In this case, no matter whether the client and server are on the same node, loopback may occur. However, if `externalTrafficPolicy: Local` is set for the Service, LB will forward them to only the node with a Server Pod. If the client and server are scheduled to different nodes through anti-affinity, no loopback will occur. Therefore, anti-affinity and `externalTrafficPolicy: Local` can avoid the loopback issue (including private network Ingress and private network LoadBalancer Service).

Does the direct LB-Pod connection of VPC-CNI have the CLB loopback issue?

TKE generally uses the Global Router network mode, and another mode is VPC-CNI (elastic network interface). Currently, direct LB-Pod connection supports only the Pods of VPC-CNI; that is, LB is directly bound to the backend

Pod rather than NodePort as the real server as shown below:



In this case, requests can bypass the NodePort instead of possibly being forwarded to any node like before. However, if the client and server are on the same node, loopback may still occur, which can be avoided through anti-affinity.

Are there any suggestions?

The anti-affinity and `externalTrafficPolicy: Local` methods are not very graceful. Generally, you should avoid accessing the CLB instance in the cluster for a service in the cluster, as the service is already in the cluster, and forwarding through CLB not only lengthens the network linkage but also may cause loopback.

When you access a service in the cluster, use the service name such as `server.prod.svc.cluster.local` as much as possible. In this way, requests will not pass CLB, and loopback will not be caused.

If your business has a coupled domain name and cannot use a Service name, you can use the rewrite plugin of coreDNS to point the domain name to a Service in the cluster. Below is the sample coreDNS configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
name: coredns
namespace: kube-system
```

```
data:
Corefile: |2-
.:53 {
rewrite name roc.oa.com server.prod.svc.cluster.local
...
```

If multiple Services share the same domain name, you can deploy an Ingress Controller (such as nginx-ingress) by yourself:

1. Use the aforementioned rewrite method to point the domain name to the self-built Ingress Controller.
2. Match the self-built Ingress with a Service according to the request location (domain name + path) and forward the request to the backend Pod. The entire linkage will not pass CLB, so loopback can be avoided.

CLB Ingress Creation Error

Last updated：2022-10-17 14:52:53

Issue

Error `E6009` was reported when I created a CLB Ingress.

```
Address:          10.8.4.119
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
TLS:
  wildcard-pagoda-2020 terminates
Rules:
  Host                      Path  Backends
  ----                      ----  --------
  cas.pagoda.com.cn
                            /   cas-api-nodeport:8080 (10.12.27.229:8080,10.12.89.129:8080,10.12.96.32:8080)
  cas-verify.pagoda.com.cn
                            /   cas-api-verify-nodeport:8080 (10.12.4.160:8080)
Annotations:                field.cattle.io/publicEndpoints:
                              [{"addresses":["10.8.4.119"],"port":80,"protocol":"HTTP","serviceName":"cas:cas-api-nodeport","ingressName":"cas:cas-internal","hostname":...
                            ingress.cloud.tencent.com/direct-access: false
                            kubernetes.io/ingress.class: qcloud
                            kubernetes.io/ingress.http-rules:
                              [{"host":"cas.pagoda.com.cn","path":"/","backend":{"serviceName":"cas-api-nodeport","servicePort":"8080"}},{"host":"cas-verify.pagoda.com....
                            kubernetes.io/ingress.https-rules:
                              [{"host":"cas.pagoda.com.cn","path":"/","backend":{"serviceName":"cas-api-nodeport","servicePort":"8080"}},{"host":"cas-verify.pagoda.com....
                            kubernetes.io/ingress.qcloud-loadbalance-id: lb-4ymlu8dc
                            kubernetes.io/ingress.rule-mix: true
                            kubernetes.io/ingress.subnetId: subnet-ewhttulw
                            qcloud_cert_id: rUWS5Gd4
Events:
  Type     Reason                  Age                From                      Message
  ----     ------                  ----               ----                      -------
  Warning  EnsureServiceFailed     15m (x61 over 74m) loadbalancer-controller   IngressError. ErrorCode: E6009 Details: UnexpectedError. OriginError: Internal error occurred: failed calling webhook "validate.nginx.ingress.kubern
etes.io": an error on the server ("") has prevented the request from succeeding
  Normal   EnsureIngressSuccess    6s (x4 over 6m3s)  loadbalancer-controller   Ingress Sync Success. RetrunCode: S2000
```

Possible Causes

ingress-nginx (Nginx Ingress community edition) versions earlier than v1.0.0 don't support validating webhook callbacks for resources of the `networking.k8s.io/v1` type. You need to remove v1 resource validation from validation CRDs.

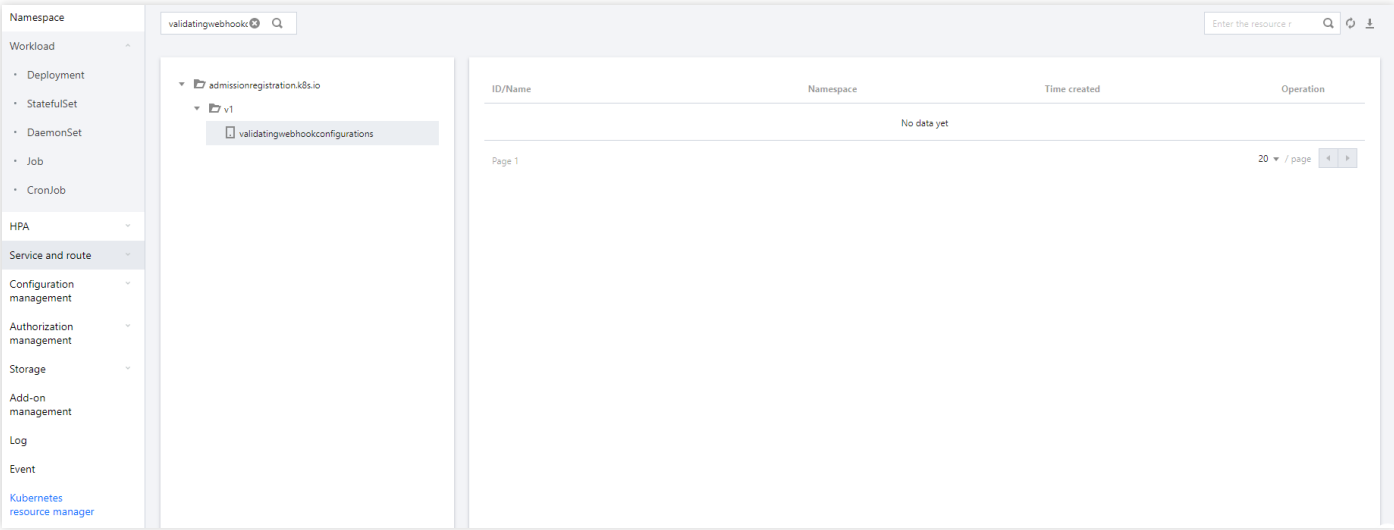
Solutions

You can solve this problem in the following ways:

Option 1. Cancel v1 resource validation

Change the `apiVersions` field in `webhooks.rules` of resources of the `validatingwebhookconfigurations` type to `v1beta1`.

1. Log in to the TKE console and select the region of your cluster.
2. On the **Cluster Management** page, click the name of the target cluster to enter its details page.
3. Select **Kubernetes resource manager** on the left sidebar and search for `validatingwebhookconfigurations` on the **Resource Type** page.



4. Select `validatingwebhookconfigurations` from the search results and click **Edit YAML** on the right of the resource object list to check whether the `apiVersions` field in `webhooks.rules` of each resource object is `v1beta1` .

```
  service:
    name: test-ingress-nginx-controller-admission
    namespace: kube-system
    path: /networking/v1beta1/ingresses
    port: 443
failurePolicy: Fail
matchPolicy: Equivalent
name: validate.nginx.ingress.kubernetes.io
namespaceSelector: {}
objectSelector: {}
rules:
- apiGroups:
  - networking.k8s.io
  apiVersions:
  - v1beta1
  operations:
  - CREATE
  - UPDATE
  resources:
  - ingresses
  scope: '*'
sideEffects: None
timeoutSeconds: 10
```

5. Upgrade the add-on. The above steps solve the resource validation problem of an existing Nginx Ingress instance. To avoid similar problems with new instances, you need to upgrade the Nginx Ingress add-on as follows:

i. On the cluster details page, select **Add-On Management** on the left sidebar.
ii. Click **Upgrade** on the right of Nginx Ingress to upgrade it to v1.1.0.

Option 2. Cancel resource validation

1. Log in to the TKE console and select the region of your cluster.
2. On the **Cluster Management** page, click the name of the target cluster to enter its details page.
3. Select **Kubernetes resource manager** on the left sidebar and search for `validatingwebhookconfigurations` on the **Resource Type** page.
4. Select `validatingwebhookconfigurations` from the search results and click **Delete** on the right of the resource object list.
5. Upgrade the add-on. The above steps solve the resource validation problem of an existing Nginx Ingress instance. To avoid similar problems with new instances, you need to upgrade the Nginx Ingress add-on as follows:
   i. On the cluster details page, select **Add-On Management** on the left sidebar.
   ii. Click **Upgrade** on the right of Nginx Ingress to upgrade it to v1.1.0.