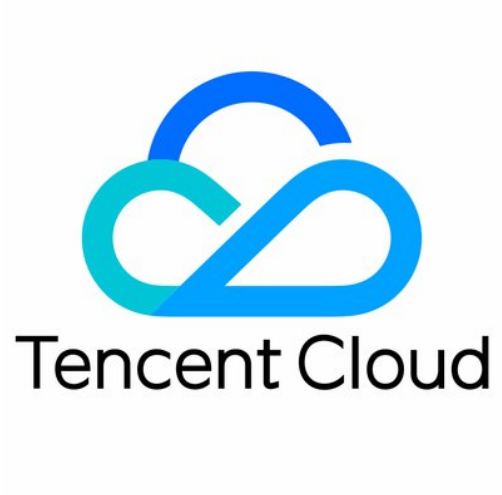
Tencent Kubernetes Engine

Fault Handling

Product Documentation



Contents

Fault Handling
Disk Full

Last updated : 2020-05-25 09:43:05

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 sandbox: rpc error: cod
e = DeadlineExceeded desc = context deadline exceeded
```

```
Warning FailedCreatePodSandBox 2m (x4307 over 16h) kubelet, 10.179.80.31 (combined from similar e
vents): Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox fo
r pod "apigateway-6dc48bf8b6-l8xrw": Error response from daemon: mkdir /var/lib/docker/aufs/mnt/1
f09d6c1c9f24e8daaea5bf33a4230de7dbc758e3b22785e8ee21e3e3d921214-init: no space left on device
```

```
Warning Failed 5m1s (x3397 over 17h) kubelet, ip-10-0-151-35.us-west-2.compute.internal (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 ¥¥¥"/s
ys¥¥¥" to rootfs ¥¥¥"/var/lib/dockerd/storage/overlay/051e985771cc69f3f699895a1dada9ef6483e912b46
a99e004af7bb4852183eb/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 id docker://apiga
teway: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 f
ailed to write checkpoint file "kubelet_internal_checkpoint": write /var/lib/kubelet/device-plugi
```

```
ns/.728425055: no space left on device, which is unexpected.
```

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 > dda02c9a7491fa797ab730c1568ba06cba74cecd4e4a82e9d90d00fa11de743c-json.log.9
# Delete log files.
```

   - 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：2020-05-25 09:43:05

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:

> 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:

> `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 24 |
MEM | tot 31.3G | free 2.1G | cache 7.6G | dirty 41.0M | buff 362.1M | slab 1.2G |
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 24 |
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 processes or threa
ds. 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 of I/O proce
sses 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 : 2020-05-25 09:43:06

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` .

  - 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=error msg="containerd: start container" error="oci runtime error: container_linux.go:247: starting container process caused ¥"process_linux.go:245: running exec setns process for init caused ¥¥¥"exit status 6¥¥¥"¥"¥n" id=5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01c
Jan 23 14:15:31 dc05 dockerd: time="2019-01-23T14:15:31.317965799+08:00" level=error msg="Create container failed with error: invalid header field value ¥"oci runtime 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] RunPodSandbox from runtime service failed: rpc error: code = 2 desc = failed to start sandbox container for pod "matchdataserver-1255064836-t4b2w": Error response from daemon: {"message":"invalid header field value ¥"oci runtime error: container_linux.go:247: starting container process caused ¥¥¥"process_linux.go:245: running 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(485fd485-1ed6-11e9-8661-0a587f8021ea)" failed: rpc error: code = 2 desc = failed to start sandbox container for pod "matchdataserver-1255064836-t4b2w": Error response from daemon: {"message":"invalid header field value ¥"oci runtime error: container_linux.go:247: starting container process caused ¥¥¥"process_linux.go:245: running exec 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(485fd485-1ed6-11e9-8661-0a587f8021ea)" failed: rpc error: code = 2 desc = failed to start sandbox container for pod "matchdataserver-1255064836-t4b2w": Error response from daemon: {"message":"invalid header field value ¥"oci runtime error: container_linux.go:247: starting container process caused ¥¥¥"process_linux.go:245: running exec 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] Error syncing pod 485fd485-1ed6-11e9-8661-0a587f8021ea ("matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)"), skipping: failed to "CreatePodSandbox" for "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)" with CreatePodSandboxError: "CreatePodSandbox for pod ¥"matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)¥" failed: rpc error: code = 2 desc = failed to start sandbox container for pod ¥"matchdataserver-1255064836-t4b2w¥": Error response from daemon: {¥"message¥":¥"invalid header field value ¥¥¥"oci runtime error: container_linux.go:247: starting container process 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] SyncLoop (PLEG): "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)", event: &pleg.PodLifecycleEvent{ID:"485fd485-1ed6-11e9-8661-0a587f8021ea", Type:"ContainerDied", Data:"5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01c"}
Jan 23 14:15:31 dc05 kubelet: W0123 14:15:31.372225 26037 pod_container_deletor.go:77] Container
```

```
"5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01c" 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 sa
ndbox for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)" 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:

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

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

Troubleshooting Method
Use Systemtap to Identify Pod Exceptions

Last updated : 2020-05-25 09:51:32

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 multiverse
deb http://ddebs.ubuntu.com/ ${codename}-updates main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-proposed main restricted universe multiverse
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:

> 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` .

> - 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://5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99cf1c945
......
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}}" 5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99cf1c945
```

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));
}
}
```

> 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(33473m38.081028267s)=>bash(33475m4.81
7798337s)=>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://5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99cf1c945
Image: ccr.ccs.tencentyun.com/library/kubedns-amd64:1.14.4
Image ID: docker-pullable://ccr.ccs.tencentyun.com/library/kubedns-amd64@sha256:40790881bbe9ef4ae
4ff7fe8b892498eecb7fe6dcc22661402f271e03f7de344
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 Troubleshooting Guide 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 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://e6746201faa1dfe7f3251b8c30d59ebf613d99715f3b800740e587e681d2a903
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-11e9-b0a7-b6655eaf0
2a6/volume-subpaths/manage-config-volume/manage/0\\\"
to rootfs \\\"/run/containerd/io.containerd.runtime.v1.linux/k8s.io/e6746201faa1dfe7f3251b8c30d59
ebf613d99715f3b800740e587e681d2a903/rootfs\\\"
at \\\"/run/containerd/io.containerd.runtime.v1.linux/k8s.io/e6746201faa1dfe7f3251b8c30d59ebf613d
99715f3b800740e587e681d2a903/rootfs/app/resources/application.properties\\\"
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 (combined from similar e
vents): Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox fo
```

```
r pod "apigateway-6dc48bf8b6-l8xrw": Error response from daemon: mkdir /var/lib/docker/aufs/mnt/1
f09d6c1c9f24e8daaea5bf33a4230de7dbc758e3b22785e8ee21e3e3d921214-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 "process_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 (combined from similar e
vents): Failed create pod sandbox: rpc error: code = Unknown desc = failed to start sandbox conta
iner for pod "nginx-7db9fccd9b-2j6dh": Error response from daemon: ttrpc: client shutting down: r
ead unix @->@/containerd-shim/moby/de2bfeefc999af42783115acca62745e6798981dff75f4148fae8c086668f6
67/shim.sock: read: connection reset by peer: unknown
Normal SandboxChanged 3m12s (x4420 over 83m) kubelet, 192.168.4.5 Pod sandbox changed, 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-d8c8b8c46-4v8tl": operation timeout
: context deadline exceeded
```

```
Warning FailedCreatePodSandBox 3s (x12 over 2m) kubelet, 10.205.8.91 Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod "lomp-ext-d8c8b8c46-4v8tl": Error response from daemon: Conflict. The container name "/k8s_POD_lomp-ext-d8c8b8c46-4v8tl_default_65046a06-f795-11e9-9bb6-b67fb7a70bad_0" is already in use by container "30aa3f5847e0ce89e9d411e76783ba14accba7eb7743e605a10a9a862a72c1e2". 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 the default value on nodes that run docker daemon with version < 1.9 or an Aufs storage 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, unlimited. (default 5)
--Registry-burst int32 Maximum size of a bursty pulls, temporarily allows pulls to burst to this number, while still not exceeding Registry-qps. Only used if --Registry-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 available: 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:

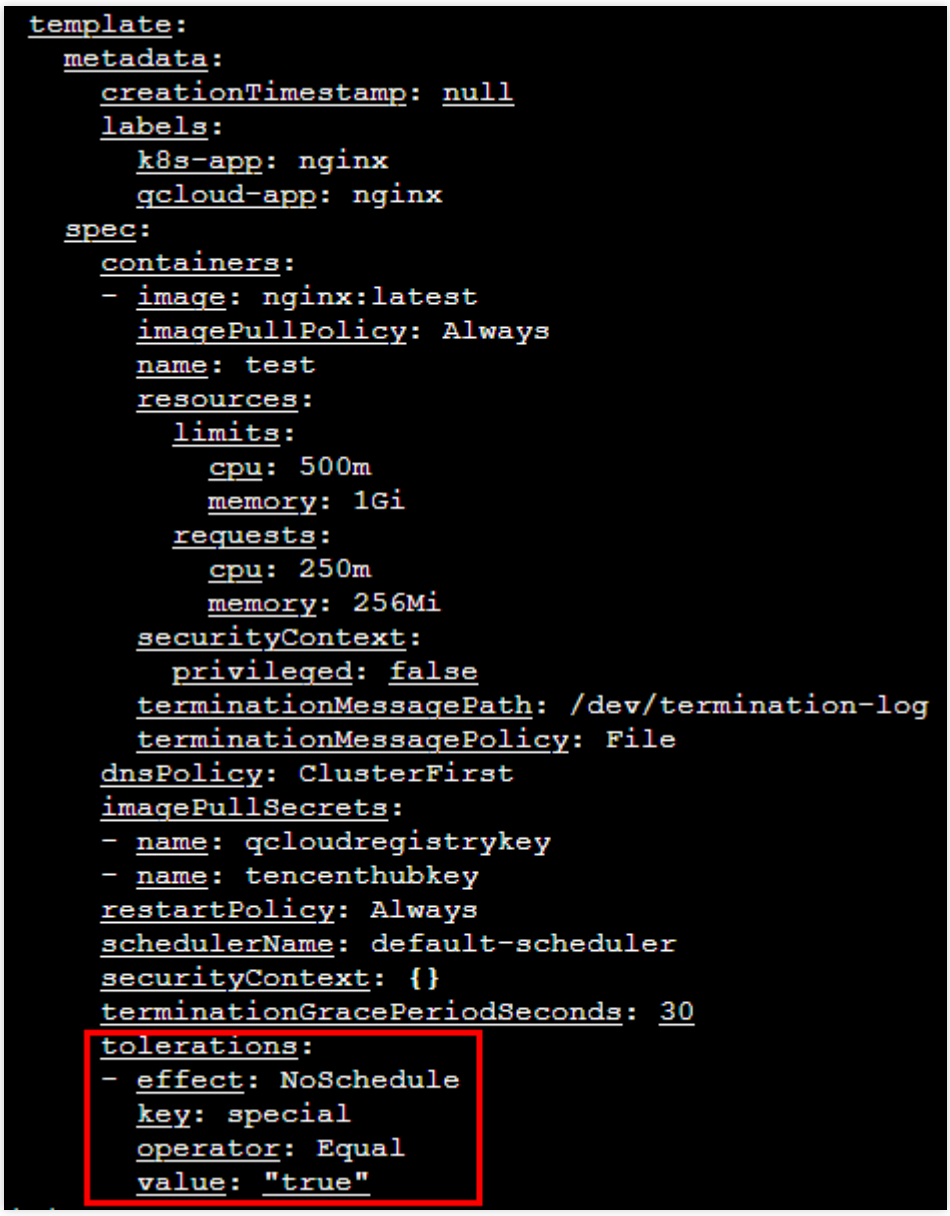
  i. Refer to Logging In to a Linux Instance in Standard Login Mode (Recommended) for instructions on how to log in to the CVM instance that contains `nginx` .
  ii. Run the following command to edit the YAML file:

  ```
  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 : 2020-05-25 09:43:07

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 id docker://apiga
teway: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 renamed, no link can be
created to this file and no data can be written to the file. Only the superuser or a process poss
essing the CAP_LINUX_IMMUTABLE capability can set or clear this attribute.
```

> 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 "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257" from runtime service failed: rpc error: code = Unknown desc = failed to remove container "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": Error response from daemon: container 19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257: driver "overlay2" failed to remove root filesystem: remove /data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db88534027546868/diff/usr/bin/bash: operation not permitted
Sep 27 14:37:21 VM_0_7_centos kubelet[14109]: E0927 14:37:21.923027 14109 kuberuntime_gc.go:126] Failed to remove container "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": rpc error: code = Unknown desc = failed to remove container "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": Error response from daemon: container 19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257: driver "overlay2" failed to remove root filesystem: remove /data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db88534027546868/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:
  i. Use the file path in the kubelet log and run the command `chattr -i <file>`, as shown below:

     ```
     chattr -i /data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db88534027546868/diff/usr/bin/bash
     ```

  ii. 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 status: 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 permission 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/54922ec8b1863bcc504f6dac41e40139047f7a84ff09175d2800100aaccbad1f/54922ec8b1863bcc504f6dac41e40139047f7a84ff09175d2800100aaccbad1f-json.log: no space 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/docker-containerd.sock task ls |grep a9a1785b81343c3ad2093ad973f4f8e52dbf54823b8bb089886c8356d4036fe0
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, optio
ns [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 96880262, win 0, l
ength 0
20:15:17.906744 IP 10.0.0.16.54132 > 172.16.2.23.8888: Flags [S], seq 1207014342, win 14600, opti
ons [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 1207014343, 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 : 2020-05-25 09:34:51

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.

> 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.

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.