Published in Dev Genius

Tony Follow
Jan 25 · 7 min read · ✨ · Listen

K8s Troubleshooting — Pod Zombie Process

K8s Troubleshooting handbook

Recently, our K8s cluster run into some Zombie process issues. Pods cannot be deleted or created, and even can't SSH into the node. We found a lots of defunct process in many Pods. The symptom in Pod looks like:

```
CPU:   0% usr   0% sys   0% nic  98% idle   0% io   0% irq   0% sirq
Load average: 0.02 0.39 0.46 4/7217 25257
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
25228 25085 root     R     1988   0%  14   0% top
   28     1 root     S    21.9g  66%  14   0% /usr/bin/java -Dspring.profiles.a
    1     0 root     S     786m   2%   9   0% ./tools/linux/env-tools
25085     0 root     S     1672   0%   5   0% sh
23278     1 root     Z        0   0%  10   0% [cat]
  157     1 root     Z        0   0%  14   0% [ssl_client]
  177     1 root     Z        0   0%  10   0% [ssl_client]
  196     1 root     Z        0   0%   0   0% [ssl_client]
...
```


```
$ ps -ef | grep defunct | wc -l
6533
```

So how to troubleshoot this kind of issue? First of all, let's understand what a `Zombie` process is.

What is Zombie Process

In short, azombie (defunct) process is a process that has completed execution, but its parent process has not yet read this process's exit code yet. So even the process is finished running, but it stays in process table. Normally zombie process doesn't use much of system resources, but it still occupies an entry in process table, which still use some memory. It can be dangerous as they can fill up the process table pretty quick.

If you want to read more about what Zombie process is, check out my article: "DevOps in Linux — Zombie Process"

Pause Container

When creating a Pod, the `kubelet` process first calls the CRI interface `RuntimeService.RunPodSandbox` to create a sandbox environment and set up the basic

operating environment such as the network. Once the Pod Sandbox is established, `kubelet` can create user containers in it. When it comes time to delete a Pod, `kubelet` will first remove the Pod Sandbox and then stop all containers inside.

The `pause` container is a container that exists in each pod, it's like a template or a parent containers from which all the new containers in the pod inherit the namespaces. The `pause` container starts, then goes to "sleep".

The `pause` container has two main responsibilities :

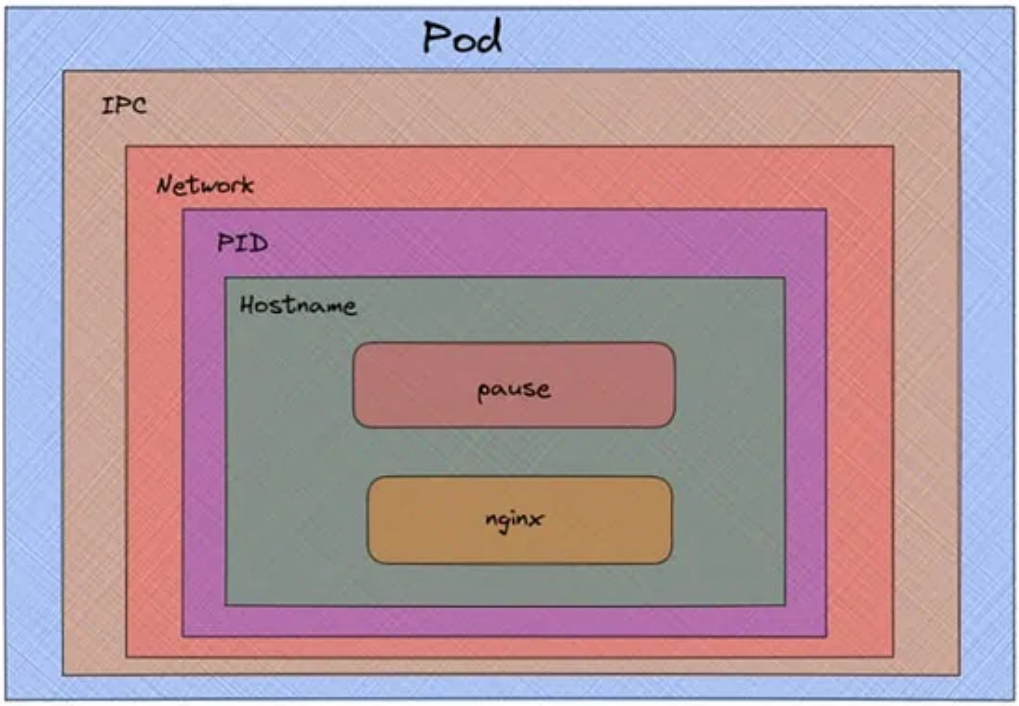
- **It serves as the basis of Linux namespace sharing in the pod.**
- **It serves as PID 1 for each ... processes (with PID namespace sharing enabled).**

The source code of pause container looks like (https://github.com/kubernetes-csi/driver-registrar/blob/master/vendor/k8s.io/kubernetes/build/pause/pause.c):

```
      }
    }
    if (getpid() != 1)
      /* Not an error because pause sees use outside of infra containers. */
      fprintf(stderr, "Warning: pause should be the first process\n");
    if (sigaction(SIGINT, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
      return 1;
    if (sigaction(SIGTERM, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
      return 2;
    if (sigaction(SIGCHLD, &(struct sigaction){.sa_handler = sigreap,
                                               .sa_flags = SA_NOCLDSTOP},
                  NULL) < 0)
      return 3;
    for (;;)
      pause();
    fprintf(stderr, "Error: infinite loop terminated\n");
    return 42;
}
```

The `pause` container basically creates a separate namespace for the Pod. While we have the `pause` container running and when we launch the actual application container, it would join the following namespaces of the `pause` container :

- **Network namespace**
- **IPC namespace**
- **PID namespace**



Pic from dev.to

This namespace sharing has the following benefits:

- Allows containers to communicate directly using the localhost.
- Allows the containers to share their inter-process communication (IPC) namespace with the other containers so they can communicate directly through shared-memory with other containers.
- Allows containers to share their process ID (PID) namespace with other containers.

Zombie Process Handling

Reaping zombies is only done by the `pause` container if you have PID namespace sharing enabled. In K8s v1.8 and above it's disabled by default unless enabled by a kubelet flag. If PID namespace sharing is not enabled then each container in a K8s pod will have its own PID 1 and each one will need to reap zombie processes itself.

If PID namespace sharing is enabled, the `/pause` process will have the PID of 1, therefore it can call the `wait` syscall after the child process has finished.

Demo

Let's build a Docker image that will generate a `zombie` process.

Dockerfile:

```
FROM python:bullseye
COPY zombie.py /root/
RUN chmod +x /root/zombie.py
ENTRYPOINT ["python", "/root/zombie.py"]
```

zombie.py:

```
import os
import subprocess

pid = os.fork()
if pid == 0:  # child
    pid2 = os.fork()
    if pid2 != 0:  # parent
        print('The zombie pid will be: {}'.format(pid2))
else:  # parent
    os.waitpid(pid, 0)
    subprocess.check_call(('ps', 'xawuf'))
```

Now let's build the image:

```
$ docker build -t <registry>/zombie:v0.0.1 .
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM python:bullseye
 ---> 63490c269128
Step 2/4 : COPY zombie.py /root/
 ---> bd4184c3ad49
...
Successfully built 295088e5c98a
```

Deploy with `shareProcessNamespace` disabled:

zombie_pod.yml:

```
apiVersion: v1
kind: Pod
metadata:
  name: zombie
spec:
  #shareProcessNamespace: true
  containers:
  - name: zombie
    image: <registry>/zombie:v0.0.1
    imagePullPolicy: Always
```

Now deploy to K8s cluster:

```
$ kubectl create -f zombie_pod.yml
pod/zombie created
```

Check zoombie process

```
$ kubectl exec -it zombie -- top
top - 19:51:52 up 66 days,  5:11,  0 users,  load average: 0.55, 0.91, 0.58
Tasks:   4 total,   1 running,   2 sleeping,   0 stopped,   1 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7847.7 total,   4136.6 free,    759.7 used,   2951.4 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   6765.3 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0   17764  14036   5636 S   0.0   0.2   0:00.15 python
    7 root      20   0   17764  10992   2580 S   0.0   0.1   0:00.00 python
    8 root      20   0       0      0      0 Z   0.0   0.0   0:00.00 python
    9 root      20   0    8940   3788   3312 R   0.0   0.0   0:00.00 top
$ kubectl get po
NAME                     READY   STATUS             RESTARTS   AGE
zombie                   0/1     CrashLoopBackOff   1          4s
```

Notice that PID 8 is a zombie process and pod is in `CrashLoopBackOff` status. A zombie process (defunct) has not been reaped since the PID 1 python process (it's parent) has not played this role. Let's also check the logs:

```
$ kubectl logs zombie
The zombie pid will be: 9
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  17748 14336 ?        Ss   20:02   0:00 python /root/z
root         9  0.0  0.0      0     0 ?        Z    20:02   0:00 [python] <defu
root        10  0.0  0.0   8652  3400 ?        R    20:02   0:00 ps xawuf
```

You can see that there is a `<defunct>` process.

Deploy with `shareProcessNamespace` enabled:

zombie_pod.yml:

```
apiVersion: v1
kind: Pod
metadata:
  name: zombie
spec:
  shareProcessNamespace: true
  containers:
  - name: zombie
    image: <registry>/zombie:v0.0.1
    imagePullPolicy: Always
  imagePullSecrets:
  - name: regcredartifactory
```

Deploy again and check if there are zombie process:

```
NAME                    READY   STATUS      RESTARTS   AGE
zombie                  0/1     Completed   1          3s
```

Notice that the `zombie` pod now is in `Completed` status. Let's check the logs

```
$ kubectl logs zombie
The zombie pid will be: 150
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       143  0.2  0.1  17764 14492 ?        Ss   19:46   0:00 python /root/z
root       151  0.0  0.0   8652  3336 ?        R    19:47   0:00  \_ ps xawuf
root         1  0.0  0.0    972     4 ?        Ss   18:51   0:00 /pause
```

We can see that the zombie process has been reaped without specifying an init process or adding one in the container. The init process now is the `pause` process (in fact it's a container).

Conclusion

Pods share many resources so it makes sense they would also share a process namespace. Some containers may expect to be isolated from others, though, so it's important to understand the differences:

- The container process no longer has PID 1. Some containers refuse to start without PID 1 (for example, containers using systemd) or run commands like kill -HUP 1 to signal the container process. In pods with a shared process namespace, kill -HUP 1 will signal the pod sandbox (/pause in the above example).
- Processes are visible to other containers in the pod. This includes all information visible in /proc, such as passwords that were passed as arguments or environment variables. These are protected only by regular Unix permissions.
- Container filesystems are visible to other containers in the pod through the /proc/$pid/root link. This makes debugging easier, but it also means that filesystem secrets are protected only by filesystem permissions.

Reference:

- https://www.back2code.me/2020/02/zombie-processes-back-in-k8s/

Kubernetes Dev Ops Cloud Computing Programming