50Kubernetes 系列（四十七）关于临时容器 - 调试Pod 的神器

Kubernetes 系列（四十七）关于临时容器 - 调试Pod 的神器

如果您关注Kubernetes的最新消息，您可能听说过Ephemeral Containers。不确定吗?不要害怕!在这篇博文中，我们将尝试解释这个在Kubernetes v1.25中稳定的新特性。

什么是临时容器?

临时容器允许我们在 Pod 中已经运行的容器的上下文中运行具有特定镜像的容器。这在调试/排除不可删除镜像或缺乏某些实用程序的镜像时非常方便，在上述情况下 `kubectl exec` 将不起作用。

```
bigbear@metalbear:~/mirrord$ kubectl exec -it py-serv-deployment-686578cbfb-9hfpw  -- sh
# tcpdump
sh: 1: tcpdump: not found
```

Let’s look at a few examples of how one might go about debugging with ephemeral containers -

I will create a new deployment using this file -

让我们看几个示例，看看如何调试临时容器—

我将使用这个文件-创建一个新的部署

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: py-serv-deployment
  labels:
    app: py-serv
spec:
  replicas: 1
  selector:
    matchLabels:
      app: py-serv
  template:
    metadata:
      labels:
        app: py-serv
    spec:
      containers:
        - name: py-serv
          image: ghcr.io/metalbear-co/mirrord-pytest:latest
          ports:
            - containerPort: 80
          env:
            - name: MIRRORD_FAKE_VAR_FIRST
              value: mirrord.is.running
            - name: MIRRORD_FAKE_VAR_SECOND
              value: "7777"

---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: py-serv
  name: py-serv
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
      nodePort: 30000
  selector:
    app: py-serv
  sessionAffinity: None
  type: NodePort
```

查询服务 -

```
bigbear@metalbear:~/mirrord$ kubectl get pods
NAME                                READY   STATUS        RESTARTS   AGE
nginx-deployment-66b6c48dd5-gmfc7   1/1     Running       0          7m7s
py-serv-deployment-686578cbfb-lb9g7   1/1     Running       0          5s
```

在 pod 中挂接一个新的容器 -

```
bigbear@metalbear:~/mirrord$ kubectl debug -it py-serv-deployment-686578cbfb-lb9g7 --image busybox
Defaulting debug container name to debugger-hthnc.
If you dont see a command prompt, try pressing enter.
/ #
```

发送请求

因为我们可以访问Pod的网络名称空间，所以我们应该能够使用curl/wget实用程序发送GET请求，这在一般镜像上是不可用的

```
/ # wget localhost:80
Connecting to localhost:80 (127.0.0.1:80)
saving to 'index.html'
index.html           100% |********************************************************************************
'index.html' saved
/ # ls
bin         dev         etc         home        index.html  proc        root        sys         tmp
/ # cat index.html
OK - GET: Request completed
/ #
```

检查网络流量

通过 `tcpdump` -

```
bigbear@metalbear:~/mirrord$ kubectl debug -it nginx-deployment-66b6c48dd5-jn5xg  -image=itsthenetwork/alp

Defaulting debug container name to debugger-mzrzj.
If you dont see a command prompt, try pressing enter.
/ # tcpdump -i any port 80
tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
17:05:43.439189 eth0  In  IP 10-244-3-10.addon-http-application-routing-nginx-ingress.kube-system.svc.clus
17:05:43.439214 eth0  Out IP nginx-deployment-66b6c48dd5-jn5xg.80 > 10-244-3-10.addon-http-application-rou
17:05:43.440148 eth0  In  IP 10-244-3-10.addon-http-application-routing-nginx-ingress.kube-system.svc.clus
17:05:43.440151 eth0  In  IP 10-244-3-10.addon-http-application-routing-nginx-ingress.kube-system.svc.clus
17:05:43.440466 eth0  Out IP nginx-deployment-66b6c48dd5-jn5xg.80 > 10-244-3-10.addon-http-application-rou
17:05:43.440630 eth0  Out IP nginx-deployment-66b6c48dd5-jn5xg.80 > 10-244-3-10.addon-http-application-rou
17:05:43.440707 eth0  Out IP nginx-deployment-66b6c48dd5-jn5xg.80 > 10-244-3-10.addon-http-application-rou
17:05:43.441058 eth0  In  IP 10-244-3-10.addon-http-application-routing-nginx-ingress.kube-system.svc.clus
17:05:43.441119 eth0  In  IP 10-244-3-10.addon-http-application-routing-nginx-ingress.kube-system.svc.clus
```

网络延迟

通过 `ping` 检查网络延迟

```
/ #  ping localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.038 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.044 ms
64 bytes from localhost (::1): icmp_seq=6 ttl=64 time=0.041 ms
64 bytes from localhost (::1): icmp_seq=7 ttl=64 time=0.037 ms
64 bytes from localhost (::1): icmp_seq=8 ttl=64 time=0.041 ms
64 bytes from localhost (::1): icmp_seq=9 ttl=64 time=0.047 ms
64 bytes from localhost (::1): icmp_seq=10 ttl=64 time=0.030 ms
64 bytes from localhost (::1): icmp_seq=11 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=12 ttl=64 time=0.040 ms
^C
--- localhost ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11266ms
rtt min/avg/max/mdev = 0.024/0.040/0.061/0.008 ms
```

它是如何工作的?

Kubernetes 将给定的镜像安排在与所选容器相同的命名空间中运行。让我们更仔细地看看哪些命名空间可用于临时容器。 执行到正在调试的pod中，列出所有名称空间，并将它们与临时容器中设置的名称空间进行比较-

`py-serv` -

```
bigbear@metalbear:~/mirrord$ kubectl exec -it py-serv-deployment-686578cbfb-lb9g7 -- sh
Defaulted container "py-serv" out of: py-serv, debugger-hthnc (ephem)
# ls -l proc/self/ns
total 0
lrwxrwxrwx 1 root root 0 Jul 19 06:17 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 ipc -> 'ipc:[4026532529]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 mnt -> 'mnt:[4026532532]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 net -> 'net:[4026532444]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 pid -> 'pid:[4026532534]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 pid_for_children -> 'pid:[4026532534]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Jul 19 06:17 uts -> 'uts:[4026532524]'
```

新创建 `debugger-tmdxk` 临时容器 -

```
bigbear@metalbear:~/mirrord$ kubectl debug -it py-serv-deployment-686578cbfb-9hfpw --share-processes --ima
Defaulting debug container name to debugger-tmdxk.
If you dont see a command prompt, try pressing enter.
/ # cd ..
/ # ls -l proc/self/ns
total 0
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 cgroup -> cgroup:[4026531835]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 ipc -> ipc:[4026532529]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 mnt -> mnt:[4026532193]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 net -> net:[4026532444]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 pid -> pid:[4026532194]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 pid_for_children -> pid:[4026532194]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 user -> user:[4026531837]
lrwxrwxrwx    1 root     root             0 Jul 19 06:11 uts -> uts:[4026532524]
```

看起来临时容器具有相同的cgroup、ipc、net、user和uts命名空间。mnt命名空间不可用是有意义的，因为临时容器和调试容器的文件系统都不同。可以通过创建pod的副本来访问pid名称空间。

但是，如果显式指定了目标容器，则临时容器可以访问pid命名空间。这意味着我们可以通过将根路径引用为 `/proc/1/root` 来访问已调试pod的文件系统

```
bigbear@metalbear:~/mirrord$ kubectl debug -it --target py-serv py-serv-deployment-686578cbfb-bh58v --imag
Targeting container "py-serv". If you don't see processes from this container it may be because the contai
Defaulting debug container name to debugger-zfd64.
If you dont see a command prompt, try pressing enter.
/ # ps
PID   USER     TIME  COMMAND
    1 root      1:49 python3 app.py
   26 root      0:00 sh
   32 root      0:00 ps
/ # cd proc/1/root
sh: getcwd
(unknown) # ls
app    boot   etc    lib    media  opt    root   sbin   sys    usr
bin    dev    home   lib64  mnt    proc   run    srv    tmp    var
```

总结

临时容器可以在你需要帮助的时候伸出援手，帮你检查网络、探测流量，非常有用！

欢迎关注我的公众号“ **云原生拓展** ”，原创技术文章第一时间推送。