



下载APP



16 | 容器网络配置（1）：容器网络不通了要怎么调试？

2020-12-21 李程远

容器实战高手课

[进入课程 >](#)**讲述：李程远**


时长 14:35 大小 13.37M



你好，我是程远。

在上一讲，我们讲了 Network Namespace 隔离了网络设备，IP 协议栈和路由表，以及防火墙规则，那容器 Network Namespace 里的参数怎么去配置，我们现在已经很清楚了。

其实对于网络配置的问题，我们还有一个最需要关心的内容，那就是容器和外面的容器或者节点是怎么通讯的，这就涉及到了容器网络接口配置的问题了。

所以这一讲呢，我们就来聊一聊，容器 Network Namespace 里如何配置网络接口，当容器网络不通的时候，我们应该怎么去做一个简单调试。

问题再现

在前面的课程里，我们一直是用 `docker run` 这个命令来启动容器的。容器启动了之后，我们也可以看到，在容器里面有一个"eth0"的网络接口，接口上也配置了一个 IP 地址。

不过呢，如果我们想从容器里访问外面的一个 IP 地址，比如说 39.106.233.176（这个是极客时间网址对应的 IP），结果就发现是不能 ping 通的。

这时我们可能会想到，到底是不是容器内出了问题，在容器里无法访问，会不会宿主机也一样不行呢？

所以我们需要验证一下，首先我们退出容器，然后在宿主机的 Network Namespace 下，再运行 `ping 39.106.233.176`，结果就会发现在宿主机上，却是可以连通这个地址的。

[复制代码](#)

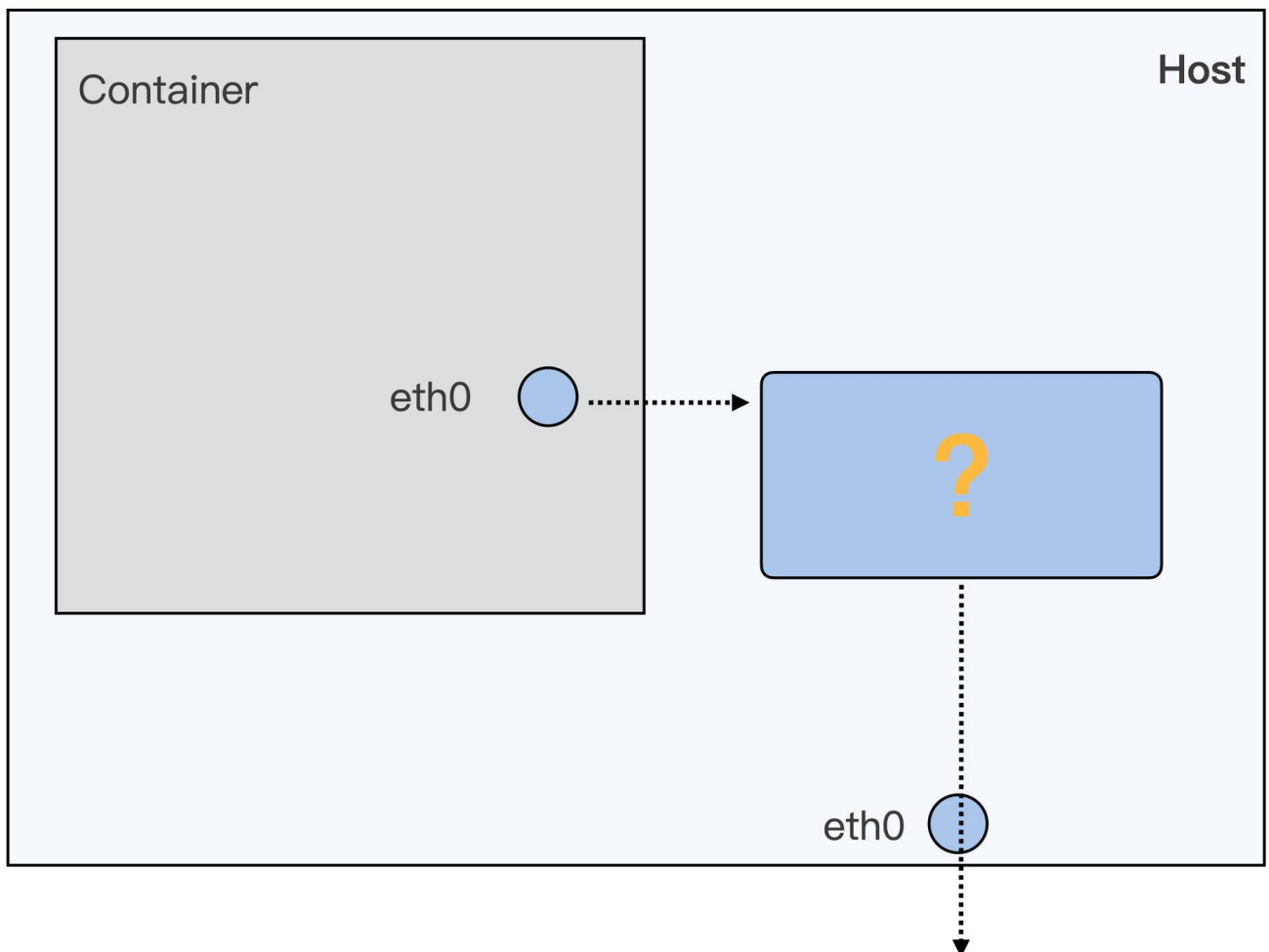
```
1 # docker run -d --name if-test centos:8.1.1911 sleep 36000
2 244d44f94dc2931626194c6fd3f99cec7b7c4bf61aafc6c702551e2c5ca2a371
3 # docker exec -it if-test bash
4
5 [root@244d44f94dc2 /]# ip addr
6 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
7     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
8     inet 127.0.0.1/8 scope host lo
9         valid_lft forever preferred_lft forever
10 808: eth0@if809: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue stat
11     link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
12     inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
13         valid_lft forever preferred_lft forever
14
15 [root@244d44f94dc2 /]# ping 39.106.233.176          ### 容器中无法ping通
16 PING 39.106.233.176 (39.106.233.176) 56(84) bytes of data.
17 ^C
18 --- 39.106.233.176 ping statistics ---
19 9 packets transmitted, 0 received, 100% packet loss, time 185ms
20
21 [root@244d44f94dc2 /]# exit          ###退出容器
22 exit
23
24 # ping 39.106.233.176          ### 宿主机上可以ping通
25 PING 39.106.233.176 (39.106.233.176) 56(84) bytes of data.
26 64 bytes from 39.106.233.176: icmp_seq=1 ttl=78 time=296 ms
27 64 bytes from 39.106.233.176: icmp_seq=2 ttl=78 time=306 ms
28 64 bytes from 39.106.233.176: icmp_seq=3 ttl=78 time=303 ms
29 ^C
30 --- 39.106.233.176 ping statistics ---
31 4 packets transmitted, 3 received, 25% packet loss, time 7ms
32 rtt min/avg/max/mdev = 296.059/301.449/305.580/4.037 ms
```

那么碰到这种容器内网络不通的问题，我们应该怎么分析调试呢？我们还是需要先来理解一下，容器 Network Namespace 里的网络接口是怎么配置的。

基本概念

在讲解容器的网络接口配置之前，我们需要先建立一个整体的认识，搞清楚容器网络接口在系统架构中处于哪个位置。

你可以看一下我给你画的这张图，图里展示的是容器有自己的 Network Namespace，eth0 是这个 Network Namespace 里的网络接口。而宿主机上也有自己的 eth0，宿主机上的 eth0 对应着真正的物理网卡，可以和外面通讯。



那你可以先想想，我们要让容器 Network Namespace 中的数据包最终发送到物理网卡上，需要完成哪些步骤呢？从图上看，我们大致可以知道应该包括这两步。

第一步，就是要让数据包从容器的 Network Namespace 发送到 Host Network Namespace 上。

第二步，数据包发到了 Host Network Namespace 之后，还要解决数据包怎么从宿主机上的 eth0 发送出去的问题。

好，整体的思路已经理清楚了，接下来我们做具体分析。我们先来看第一步，怎么让数据包从容器的 Network Namespace 发送到 Host Network Namespace 上面。

你可以查看一下 [🔗 Docker 网络的文档](#) 或者 [🔗 Kubernetes 网络的文档](#)，这些文档里面介绍了很多种容器网络配置的方式。

不过对于容器从自己的 Network Namespace 连接到 Host Network Namespace 的方法，一般来说就只有两类设备接口：一类是 [🔗 veth](#)，另外一类是 macvlan/ipvlan。

在这些方法中，我们使用最多的就是 veth 的方式，用 Docker 启动的容器缺省的网络接口用的也是这个 veth。既然它这么常见，所以我们就用 veth 作为例子来详细讲解。至于另外一类 macvlan/ipvlan 的方式，我们在下一讲里会讲到。

那什么是 veth 呢？为了方便你更好地理解，我们先来模拟一下 Docker 为容器建立 eth0 网络接口的过程，动手操作一下，这样呢，你就可以很快明白什么是 veth 了。

对于这个模拟操作呢，我们主要用到的是 [🔗 ip netns](#) 这个命令，通过它来对 Network Namespace 做操作。

首先，我们先启动一个不带网络配置的容器，和我们之前的命令比较，主要是多加上了"--network none"参数。我们可以看到，这样在启动的容器中，Network Namespace 里就只有 loopback 一个网络设备，而没有了 eth0 网络设备了。


 复制代码

```
1 # docker run -d --name if-test --network none centos:8.1.1911 sleep 36000
2 cf3d3105b11512658a025f5b401a09c888ed3495205f31e0a0d78a2036729472
3 # docker exec -it if-test ip addr
4 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
5     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
6     inet 127.0.0.1/8 scope host lo
```



```
7 valid 1ft forever preferred 1ft forever
```

完成刚才的设置以后，我们就在这个容器的 Network Namespace 里建立 veth，你可以执行一下后面的这个脚本。

 复制代码

```
1 pid=$(ps -ef | grep "sleep 36000" | grep -v grep | awk '{print $2}')
2 echo $pid
3 ln -s /proc/$pid/ns/net /var/run/netns/$pid
4
5 # Create a pair of veth interfaces
6 ip link add name veth_host type veth peer name veth_container
7 # Put one of them in the new net ns
8 ip link set veth_container netns $pid
9
10 # In the container, setup veth_container
11 ip netns exec $pid ip link set veth_container name eth0
12 ip netns exec $pid ip addr add 172.17.1.2/16 dev eth0
13 ip netns exec $pid ip link set eth0 up
14 ip netns exec $pid ip route add default via 172.17.0.1
15
16 # In the host, set veth_host up
17 ip link set veth_host up
```

我在这里解释一下，这个 veth 的建立过程是什么样的。

首先呢，我们先找到这个容器里运行的进程"sleep 36000"的 pid，通过"/proc/\$pid/ns/net"这个文件得到 Network Namespace 的 ID，这个 Network Namespace ID 既是这个进程的，也同时属于这个容器。

然后我们在"/var/run/netns/"的目录下建立一个符号链接，指向这个容器的 Network Namespace。完成这步操作之后，在后面的"ip netns"操作里，就可以用 pid 的值作为这个容器的 Network Namesapce 的标识了。

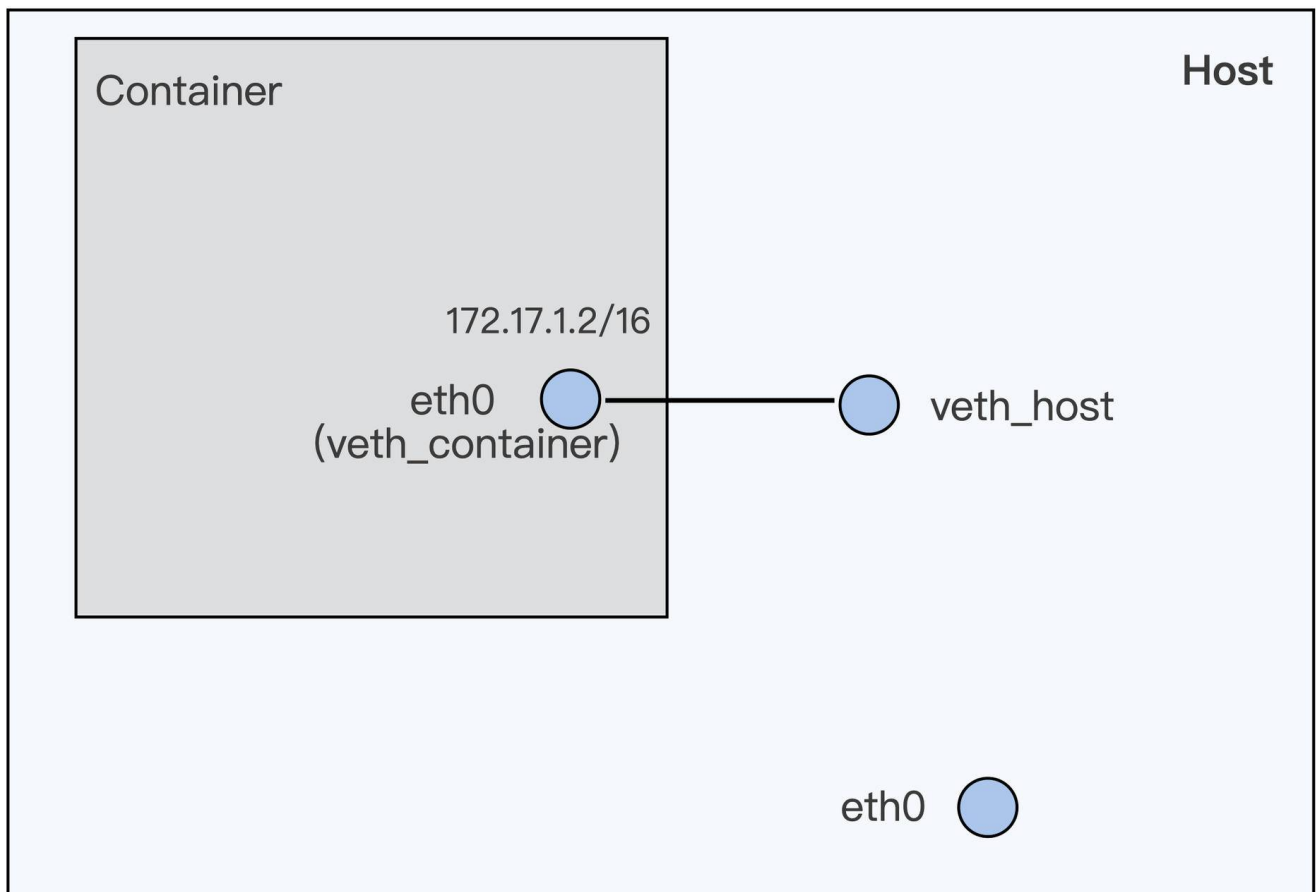
接下来呢，我们用 ip link 命令来建立一对 veth 的虚拟设备接口，分别是 veth_container 和 veth_host。从名字就可以看出来，veth_container 这个接口会被放在容器 Network Namespace 里，而 veth_host 会放在宿主机的 Host Network Namespace。

所以我们后面的命令也很好理解了，就是用 `ip link set veth_container netns $pid` 把 `veth_container` 这个接口放入到容器的 Network Namespace 中。

再然后我们要把 `veth_container` 重新命名为 `eth0`，因为这时候接口已经在容器的 Network Namespace 里了，`eth0` 就不会和宿主机上的 `eth0` 冲突了。


最后对容器内的 `eth0`，我们还要做基本的网络 IP 和缺省路由配置。因为 `veth_host` 已经在宿主机的 Host Network Namespace 了，就不需要我们做什么了，这时我们只需要 `up` 一下这个接口就可以了。

那刚才这些操作完成以后，我们就建立了一对 veth 虚拟设备接口。我给你画了一张示意图，图里直观展示了这对接口在容器和宿主机上的位置。



现在，我们再来看看 veth 的定义了，其实它也很简单。veth 就是一个虚拟的网络设备，一般都是成对创建，而且这对设备是相互连接的。当每个设备在不同的 Network Namespaces 的时候，Namespace 之间就可以用这对 veth 设备来进行网络通讯了。

比如说, 你可以执行下面的这段代码, 试试在 `veth_host` 加上一个 IP, `172.17.1.1/16`, 然后从容器里就可以 `ping` 通这个 IP 了。这也证明了从容器到宿主机可以利用这对 `veth` 接口来通讯了。

 复制代码

```
1 # ip addr add 172.17.1.1/16 dev veth_host
2 # docker exec -it if-test ping 172.17.1.1
3 PING 172.17.1.1 (172.17.1.1) 56(84) bytes of data.
4 64 bytes from 172.17.1.1: icmp_seq=1 ttl=64 time=0.073 ms
5 64 bytes from 172.17.1.1: icmp_seq=2 ttl=64 time=0.092 ms
6 ^C
7 --- 172.17.1.1 ping statistics ---
8 2 packets transmitted, 2 received, 0% packet loss, time 30ms
9 rtt min/avg/max/mdev = 0.073/0.082/0.092/0.013 ms
```

好了, 这样我们完成了第一步, 通过一对 `veth` 虚拟设备, 可以让数据包从容器的 `Network Namespace` 发送到 `Host Network Namespace` 上。

那下面我们再来看第二步, 数据包到了 `Host Network Namespace` 之后呢, 怎么把它从宿主机上的 `eth0` 发送出去?

其实这一步呢, 就是一个普通 Linux 节点上数据包转发的问题了。这里我们解决问题的方法有很多种, 比如说用 `nat` 来做个转发, 或者建立 `Overlay` 网络发送, 也可以通过配置 `proxy arp` 加路由的方法来实现。

因为考虑到网络环境的配置, 同时 `Docker` 缺省使用的是 **bridge + nat** 的转发方式, 那我们就在刚才讲的第一步基础上, 再手动实现一下 `bridge+nat` 的转发方式。对于其他的配置方法, 你可以看一下 `Docker` 或者 `Kubernetes` 相关的文档。

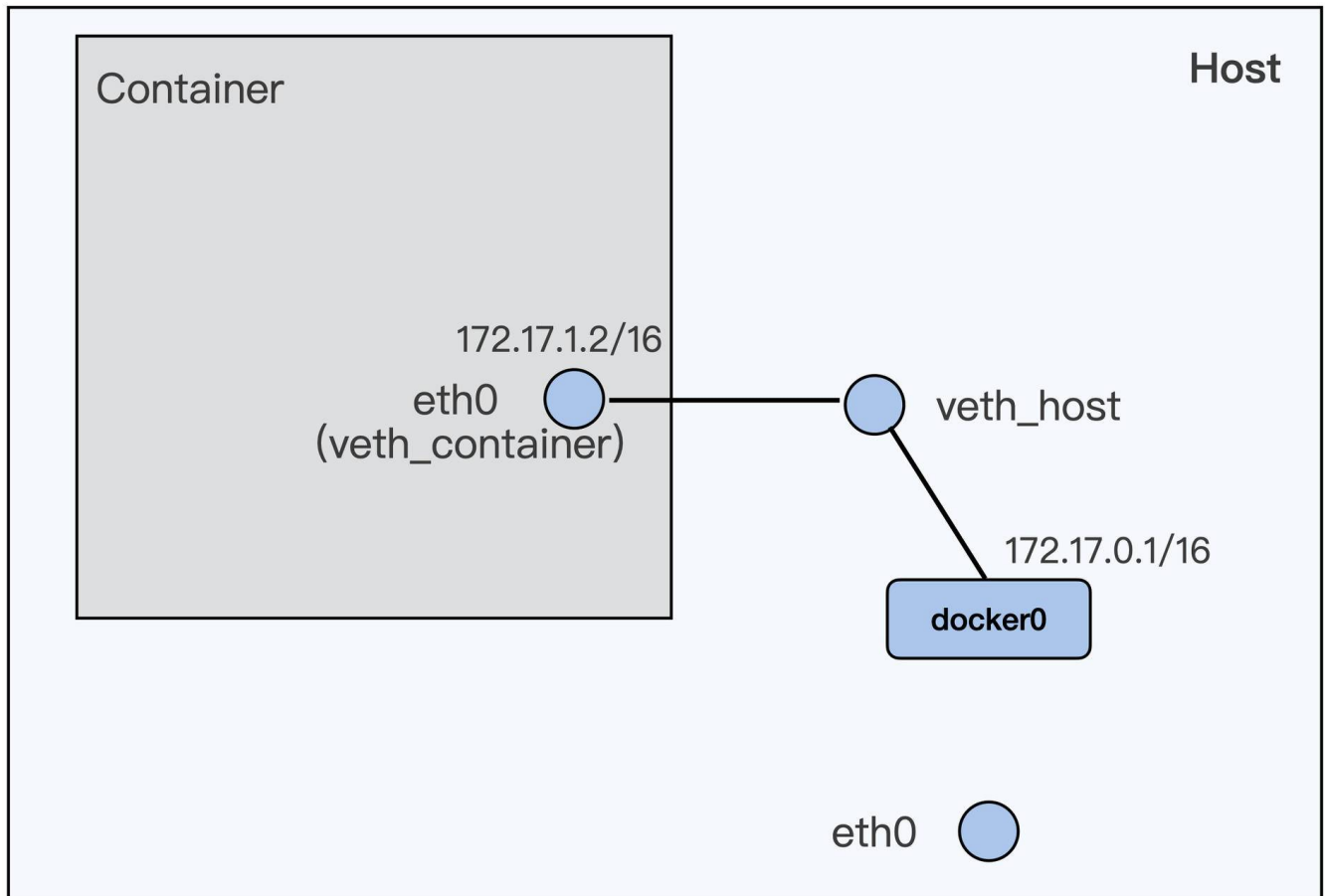
`Docker` 程序在节点上安装完之后, 就会自动建立了一个 `docker0` 的 `bridge interface`。所以我们只需要把第一步中建立的 `veth_host` 这个设备, 接入到 `docker0` 这个 `bridge` 上。

这里我要提醒你注意一下, 如果之前你在 `veth_host` 上设置了 IP 的, 就需先运行一下 "`ip addr delete 172.17.1.1/16 dev veth_host`", 把 IP 从 `veth_host` 上删除。

```
1 # ip addr delete 172.17.1.1/16 dev veth_host
2 ip link set veth_host master docker0
```

[复制代码](#)

这个命令执行完之后，容器和宿主机的网络配置就会发生变化，这种配置是什么样呢？你可以参考一下面这张图的描述。



从这张示意图中，我们可以看出来，容器和 `docker0` 组成了一个子网，`docker0` 上的 IP 就是这个子网的网关 IP。

如果我们要让子网通过宿主机上 `eth0` 去访问外网的话，那么加上 `iptables` 的规则就可以了，也就是下面这条规则。

```
1 iptables -P FORWARD ACCEPT
```

[复制代码](#)

好了，进行到这里，我们通过 `bridge+nat` 的配置，似乎已经完成了第二步——让数据从宿主机的 `eth0` 发送出去。

那么我们这样配置, 真的可以让容器里发送数据包到外网了吗? 这需要我们做个测试, 再重新尝试下这一讲开始的操作, 从容器里 ping 外网的 IP, 这时候, 你会发现还是 ping 不通。

其实呢, 做到这一步, 我们通过自己的逐步操作呢, 重现了这一讲了最开始的问题。


解决问题

既然现在我们清楚了, 在这个节点上容器和宿主机上的网络配置是怎么一回事。那么要调试这个问题呢, 也有了思路, 关键就是找到数据包传到哪个环节时发生了中断。

那最直接的方法呢, 就是在容器中继续 ping 外网的 IP 39.106.233.176, 然后在容器的 eth0 (veth_container), 容器外的 veth_host, docker0, 宿主机的 eth0 这一条数据包的路径上运行 tcpdump。


这样就可以查到, 到底在哪个设备接口上没有收到 ping 的 icmp 包。我把 tcpdump 运行的结果我列到了下面。

容器的 eth0:

 复制代码

```
1 # ip netns exec $pid tcpdump -i eth0 host 39.106.233.176 -nn
2 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
3 listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
4 00:47:29.934294 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
5 00:47:30.934766 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
6 00:47:31.958875 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
```

veth_host:

 复制代码


```
1 # tcpdump -i veth_host host 39.106.233.176 -nn
2 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
3 listening on veth_host, link-type EN10MB (Ethernet), capture size 262144 bytes
4 00:48:01.654720 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
5 00:48:02.678752 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
6 00:48:03.702827 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
```

docker0:

 复制代码

```
1 # tcpdump -i docker0 host 39.106.233.176 -nn
2 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
3 listening on docker0, link-type EN10MB (Ethernet), capture size 262144 bytes
4 00:48:20.086841 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
5 00:48:21.110765 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
6 00:48:22.134839 IP 172.17.1.2 > 39.106.233.176: ICMP echo request, id 71, seq
```


host eth0:

 复制代码

```
1 # tcpdump -i eth0 host 39.106.233.176 -nn
2 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
3 listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
4 ^C
5 0 packets captured
6 0 packets received by filter
7 0 packets dropped by kernel
```

通过上面的输出结果，我们发现 icmp 包到达了 docker0，但是没有到达宿主机上的 eth0。

因为我们已经配置了 iptables nat 的转发，这个也可以通过查看 iptables 的 nat 表确认一下，是没有问题的，具体的操作命令如下：


 复制代码

```
1 # iptables -L -t nat
2 Chain PREROUTING (policy ACCEPT)
3 target    prot opt source                destination
4 DOCKER    all  --  anywhere              anywhere           ADDRTYPE match d
5
6 Chain INPUT (policy ACCEPT)
7 target    prot opt source                destination
8
9 Chain POSTROUTING (policy ACCEPT)
10 target    prot opt source                destination
11 MASQUERADE all  --  172.17.0.0/16         anywhere
12
13 Chain OUTPUT (policy ACCEPT)
14 target    prot opt source                destination
```

```
15 DOCKER      all  --  anywhere          !127.0.0.0/8          ADDRTYPE match d
16
17 Chain DOCKER (2 references)
18 target      prot opt source          destination
19 RETURN      all  --  anywhere          anywhere
```

那么会是什么问题呢？因为这里需要做两个网络设备接口之间的数据包转发，也就是从 docker0 把数据包转发到 eth0 上，你可能想到了 Linux 协议栈里的一个常用参数 ip_forward。

我们可以看一下，它的值是 0，当我们把它改成 1 之后，那么我们就可以从容器中 ping 通外网 39.106.233.176 这个 IP 了！

 复制代码

```
1 # cat /proc/sys/net/ipv4/ip_forward
2 0
3 # echo 1 > /proc/sys/net/ipv4/ip_forward
4
5 # docker exec -it if-test ping 39.106.233.176
6 PING 39.106.233.176 (39.106.233.176) 56(84) bytes of data.
7 64 bytes from 39.106.233.176: icmp_seq=1 ttl=77 time=359 ms
8 64 bytes from 39.106.233.176: icmp_seq=2 ttl=77 time=346 ms
9 ^C
10 --- 39.106.233.176 ping statistics ---
11 2 packets transmitted, 2 received, 0% packet loss, time 1ms
12 rtt min/avg/max/mdev = 345.889/352.482/359.075/6.593 ms
```

重点小结

这一讲，我们主要解决的问题是如何给容器配置网络接口，让容器可以和外面通讯；同时我们还学习了当容器网络不通的时候，我们应该怎么来做一个简单调试。

解决容器与外界通讯的问题呢，一共需要完成两步。第一步是，怎么让数据包从容器的 Network Namespace 发送到 Host Network Namespace 上；第二步，数据包到了 Host Network Namespace 之后，还需要让它可以从宿主机的 eth0 发送出去。

我们想让数据从容器 Network Namespace 发送到 Host Network Namespace，可以用配置一对 veth 虚拟网络设备的方法实现。而让数据包从宿主机的 eth0 发送出去，就用可 bridge+nat 的方式完成。

这里我讲的是最基本的一种配置，但它也是很常用的一个网络配置。针对其他不同需要，容器网络还有很多种。那你学习完这一讲，了解了基本的概念和操作之后呢，还可以查看更多的网上资料，学习不同的网络配置。

遇到容器中网络不通的情况，我们先要理解自己的容器以及容器在宿主机上的配置，通过对主要设备上做 tcpdump 可以找到具体在哪一步数据包停止了转发。

然后我们结合内核网络配置参数，路由表信息，防火墙规则，一般都可以定位出根本原因，最终解决这种网络完全不通的问题。

但是如果是网络偶尔丢包的问题，这个就需要用到其他的一些工具来做分析了，这个我们会在之后的章节做讲解。

思考题

我们这一讲的例子呢，实现了从容器访问外面的 IP。那么如果要实现节点外的程序来访问容器的 IP，我们应该怎么配置网络呢？

欢迎你在留言区分享你的思考和问题。如果这篇文章对你有启发，也欢迎分享给你的朋友，一起学习进步。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | 容器网络：我修改了 /proc/sys/net 下的参数，为什么在容器中不起效？

精选留言 (2)

写留言



良凯尔

2020-12-21

实现节点外的程序来访问容器：

```
iptables -t nat -A PREROUTING -d 【宿主机ip】 -p tcp -m tcp --dport 【宿主机映射端口】 -j DNAT --to-destination 【容器ip】:【容器端口】
```

...

展开 ✓



👍 2

**谢哈哈**

2020-12-21

1，宿主机上配置容器网段的路由

2，DNAT，在nat表的PREROUTING链做好包伪装到达容器

展开 ✓



👍 1