Docker(二) 网络原理

前言

上一次我们讲述了 docker 文件系统中的一些原理。发现了容器的一些猫腻，
我们进入容器看到的目录其实是联合文件系统做出来的伪操作系统。 docker 使用这种方式实现了目录和文件系统的隔离。
但要让一个容器成型光有文件系统的隔离是不够的， 我们进入容器后运行一个 ps aux
的时候会看到其他容器或者宿主机启动的进程，运行一个 ifconfig 的时候会看到其他容器的网卡。所以除此之外还需要进程和网络的隔离。
这样才能让一个容器看起来就是一个真正的操作系统一样。

namespace

namespace(名称空间)，我们可以理解为这是 linux
为了隔离网络，进程，文件等等搞出来个一个机制，所以它就有网络名称空间，进程名称空间，挂载文件系统名称空间等。
不同名称空间是隔离的，互相无法通信，也无法感知。 也就是说如果在你名称空间 A 中创建了一个进程，那在名称空间 B
中你是看不到这个进程的，也无法与这个进程进行网络通信。不同的名称空间之间的关系就好像是平行宇宙一样。
每个名称空间都可以拥有自己的网络设备，路由表，Iptables/Netfilter 设置。
这样，通过上一篇我们讲到的联合文件系统和名称空间，docker
就能实现对一个操作系统的模拟以及容器之间的隔离，所以我们在进入容器的时候也就会发现我们看不到宿主机和其他容器的任何东西
(进程，网络设备，文件)。

网络名称空间操作

我们来模拟一下 docker 对名称空间的操作。

```
sudo ip netns add Container_ns1
sudo ip netns add Container_ns2

$ sudo ip netns list
Container_ns2
Container_ns1
```

我们探索一下新创建的 ns 的网络空间 (通过 ip netns exec 命令可以在特定 ns 的内部执行相关程序

```
$ sudo ip netns exec Container_ns1 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

$ sudo ip netns exec Container_ns2 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

$ sudo ip netns exec Container_ns2 ip route
```

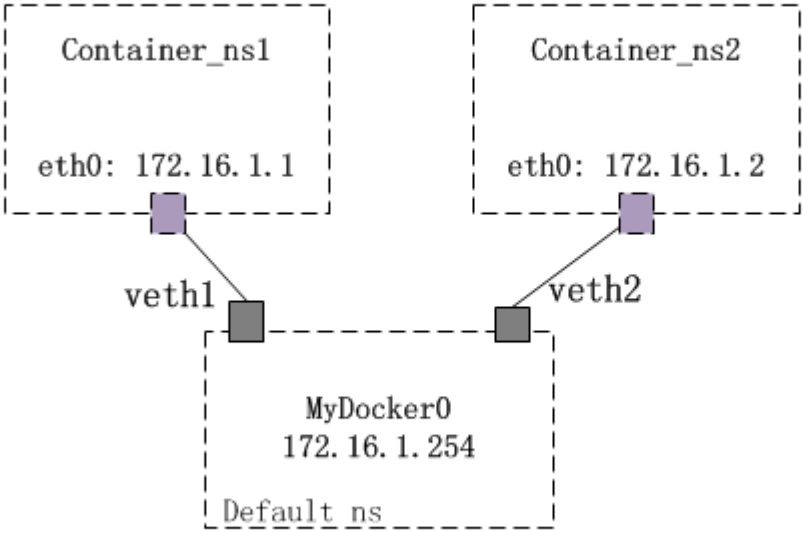
可以看到，新建的 ns 的网络设备只有一个 loopback 口，并且路由表为空。而且他们双方都看不到对方的
loopback，你也看不到宿主机（也就是 root 名称空间）的任何进程和网络设备。所以说，我们的宿主机 (root), Container_ns1 和
Container_ns2 三者之间是隔离的。

Veth 设备对

当我们有了名称空间后可以实现网络和进程的隔离。但我们是有跟其他容器和外界网络进行通讯的需求的，那这个怎么办呢？ 我们再来看看
linux 基础网络设备中的 Veth(虚拟网卡设备对)。 引入 Veth
设备对是为了在不同的网络名称空间之间进行通信，利用它可以直接将两个名称空间连接起来。 由于要连接两个网络名称空间，所以 Veth
设备都是成对出现的，很像一对网卡，并且中间有一根相连的线。 我们将其中一端称为另一端的 peer。 Veth
设备对的特性是无论像任何一端发送网络请求，它都会转发给他的 peer 进行接收。 所以如果我们把一对 veth
设备放到两个名称空间中，那么名称空间 A 就可以通过向自己的 Veth 设备发送网络请求的方式与名称空间 B 进行通信了。

linux 网桥

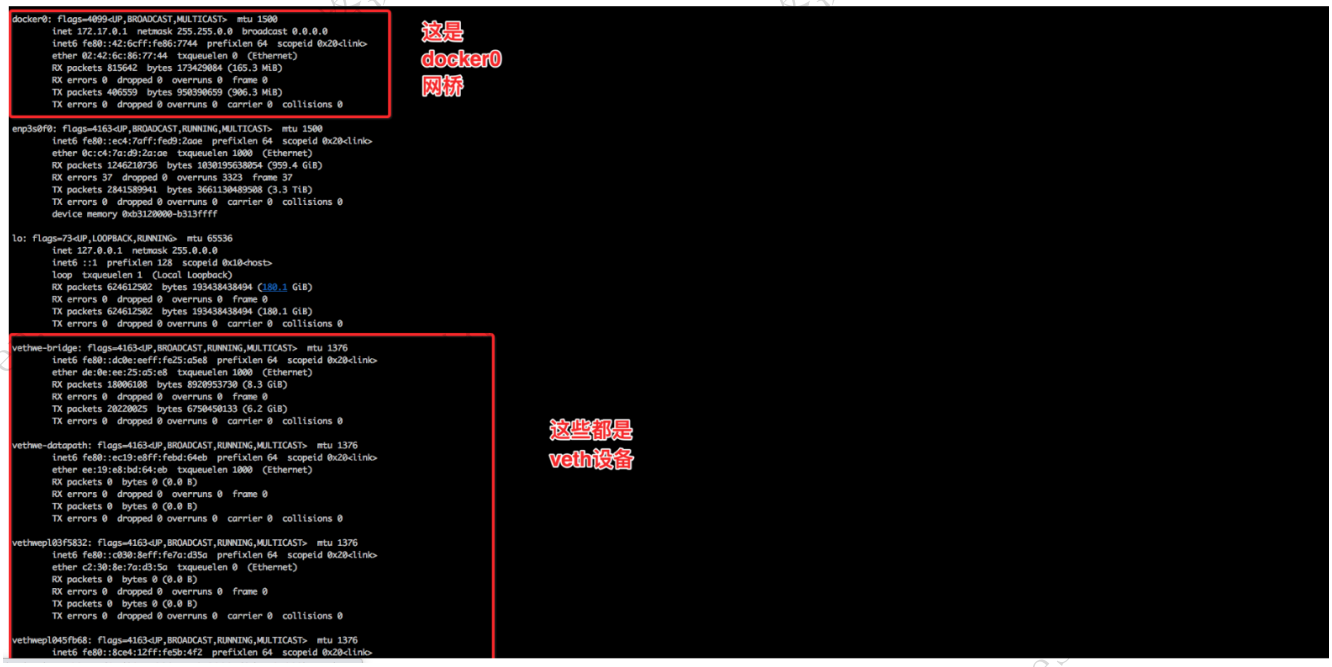
通过 Veth 我们能让两个不同的名称空间进行网络通信，但这还不够。
如果说一个容器对应一个名称空间的话，那么我们需要把无限多个容器网络打通仅仅靠 Veth 就不够了。 这时候就出现了 linux 网桥。briage
是 Linux 提供的一种虚拟网络设备之一。其工作方式非常类似于物理的网络交换机设备。Linux Bridge
可以工作在二层，也可以工作在三层，默认工作在二层。工作在二层时，可以在同一网络的不同主机间转发以太网报文；一旦你给一个 Linux
Bridge 分配了 IP 地址，也就开启了该 Bridge 的三层工作模式。在 Linux 下，你可以用 iproute2 工具包或 brctl 命令对 Linux bridge
进行管理。所以结合着名称空间，网桥和 Veth。我们就有如下的网络模型。



这也就是 docker 网络模型的略缩图了，我们有 Container_ns1 和 Container_ns2 两个名称空间，分别对应两个容器。
他们之间的网络是隔离的。 那么为了能让两个容器之间进行网络通信，让所有容器和宿主机进行网络通信。 我们采取在宿主机也就是 root 名称空间中创建一个网桥的方式。同时为每个名称空间创建一对 veth 设备， veth 设备的一端放在容器名称空间中，另一端放在 root 名称空间中并挂在之前创建的网桥上。 这样容器可以通过 veth
设备对与宿主机的网桥通信，网桥会把网络请求转发到宿主机的网卡上与宿主机通信，同时也会把请求转发给网桥上其他容器的 veth 设备上去以保证所有容器之间的网络通信。 这样就构成了一个 briage 模式的容器网络模型。

阶段性总结

如果你有一台装着 docker 的服务器，你可以登录上去使用 ip 命令查看一下，一定有一个名字叫做 docker0 的网桥。 如果你运行一个 ifconfig 命令，你会发现有很多随机名称的 veth 设备。因为每启动一个容器就会创建相应的 veth 设备对。 只不过你看不到 veth 设备对的 peer 了，因为它们都在容器的名称空间里，你再宿主机上 (root 名称空间) 是看不到的。如下图。



再说什么是容器

那么我们再来说说到底什么是容器吧。 通过上篇文章我们知道了容器是一个伪操作系统，实际上我们是在宿主机上运行的，只不过 docker

通过了一些手段来让我们以为是在一个完整的操作系统上。 那么这些手段主要就是两个：

- 通过联合文件系统，模拟操作系统的目录并展示给用户，修改各种环境变量，以营造出文件系统和目录的隔离。我们以为运行了不同的操作系统，实际上是镜像层安装了各种操作系统的命令脚本，把环境变量一改。
  实际上我们是用宿主机的内核，运行着镜像层安装的各种操作系统版本的命令罢了。
- 通过名称空间，veth 和
  briage，创造出网络和进程级别的隔离，进一步的伪造一个操作系统。同时通过今天描述的内容打通容器网络。

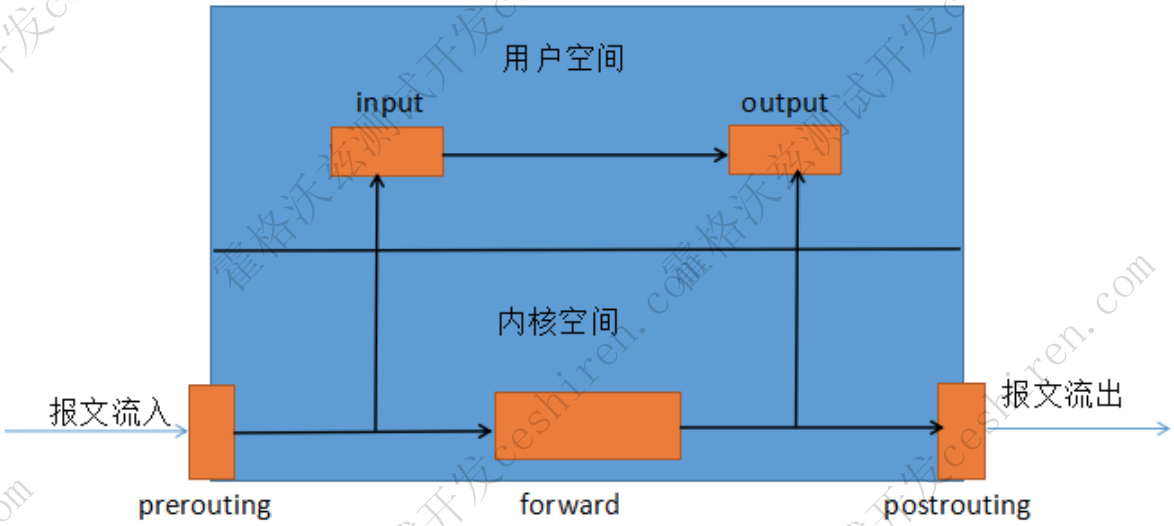
通过上面两个方法，我们的每个容器就都看上去是一个独立的操作系统了。 但我们现在知道了原理，就能明白一件事情：实际上我们都是在宿主机上跑。所以我们之前总能听说 docker 的容器之间是共享内核的
(废话都是在一个操作系统上跑的用的当然是一个内核，只是用的不同的命令而已)， docker 相比虚拟机是轻量级的
(废话虚拟机那是启动一个完整的操作系统，容器充其量就一进程)，docker 启动快速几乎是秒级的
(废话，东西都装在镜像层里了，容器几乎啥也不用干当然快，你让虚拟机把所有东西实现都装好也一样快)

端口映射

上面的网络模型可以让容器之间，容器和宿主机之前互相通信。 但是它无法做到让外界访问容器，为什么呢？ 我们知道不论 A 类，B 类还是 C 类的 IP 地址都预留一段私有 IP 地址端，这一段 ip 地址是不进路由规则的，而 docker 给容器分配的 ip 地址就是私有 IP 地址。 也就是说 docker 为容器建立了一个私有的局域网络。 这个道理很简单，我们自己在家上网的时候用的也是私有 ip
地址，你再公司是连不上自己的电脑的。那我们怎么访问容器，容器怎么对外暴露服务？答案是通过 iptables。

iptables

我就不详解 iptables 了，就简单说说原理吧。iptables 是一个工具，可以创造规则间接的控制内核空间中的 netfilter 的工具。
它可以创建规则来阻止某些特定的请求来形成防火墙规则。也可以通过 SNAT 和 DNAT
技术修改网络请求的源地址和目的地址来达到网络转发的功能。 而我们利用的就是后者来达到我们与容器通信的目的。 我们看上面的图。
在我们的 linux 网络协议栈中留了 5 个口。 分别是 input(数据流入)，output(数据流出)，forward(数据转发)，prerouting(路由前) 和 postrouting(路由后)。 我们在做防火墙规则的时候主要利用的就是 input 和 output。而使用 DNAT 和 SNAT 做转发的时候，主要用的就是 prerouting 和 postrouting。 那么 docker 最主要利用的就是 DNAT 来修改目的地址。假如说我们定义一条规则，凡是访问宿主机 ip 的 80 端口的请求全部转发到容器的 8080 端口。 这样当 ip 报文到达网络协议栈中的 prerouting 的时候，我们的系统就会把 ip
报文拆开，将目的地址修改为容器的 ip 和端口号。 再通过 forward 链传递到 docker0 网桥上 (前提是你的 linux 系统开启了 ipv4_forward，意思是支持转发，记住安装 docker 的话一定要打开这个开关)，docker0 再传递到容器内。如下图



用过 docker run 命令的小伙伴一定知道-p 这个参数-- 端口映射，把容器暴露的端口映射到宿主机的端口上。这样我们就可以通过访问宿主机 ip+ 端口号的方式访问容器了。 那么这么做的原理就是 iptables 通过 DNAT 修改了 ip 报文中的目的地址。

troubleshooting

如果你的容器网络出现问题，请按照以下步骤来排查。

- 使用 docker exec 命令进入容器，查看服务是否启动，netstat 查看端口是否监听到，查看 DNS，网关是否能 ping 通。
- 查看宿主机系统的 ipv4_forward 是否打开
- 查看系统是否安装了 iptables
- 查看 iptables 规则是不是有防火墙规则把你的容器网络堵死了
  (大概率事件，测试环境总有人去装东西，不知道哪个软件就自带一堆规则，我们的 k8s 网络曾经被 iptables 坑了好几次)

总结

OK，今天讲的网络原理其实就是 docker 的 4 中网络模式中的 briage 模式,其他的模式虽然不一样但是只有一些略微的差别，都是利用 namespace，veth 和 briage 来做一些花样，剩余的三种网络模式我们也会在下一篇的 docker 网络解决方案中讲解。