链滴

从 Linux 网络虚拟化探究容器网络

作者：Gakkiyomi2019

原文链接：https://ld246.com/article/1681456619163

来源网站：链滴

容器的本质是一个特殊的进程，特殊在为期创建了 Namespace 隔离运行环境，并用 CGroup 为其制资源开销。借助这两个技术我们可以成功实现应用容器化，但如何让多个容器在网络环境相互通信以及访问外部网络，或者让外部网络访问特定容器等问题，则还需要利用一些Linux网络虚拟化技术。

在Linux提供的众多Namespace中，我们可以其中Network Namespace 来给容器配置独立的网络图。

现在让我们从linux提供的一些网络虚拟化技术来实现一个简易版的"docker"。

网络命名空间隔离

通过一个小实验demo来体验网络命名空间

通过使用命令 readlink proc/$$/ns/net来查看宿主机的网络空间

```
[root@tsy7461 ~]# readlink /proc/$$/ns/net
net:[4026531956]
[root@tsy7461 ~]#
```

我们也可以创建Network Namespace，通过ip netns 工具来创建

```
[root@tsy7461 ~]# ip netns add fangcong1
[root@tsy7461 ~]# ip netns add fangcong2
```

```
[root@tsy7461 ~]# ip netns list
fangcong2
fangcong1
[root@tsy7461 ~]#
```

在这两个网络命名空间上创建两个bash容器 进行观察, 发现 网络空间的值改变了

```
[root@tsy7461 ~]# ip netns exec fangcong1 /bin/bash --rcfile <(echo "PS1=\"fangcong1> \"")
fangcong1> readlink /proc/$$/ns/net
net:[4026532550]
fangcong1>
```

查看接口

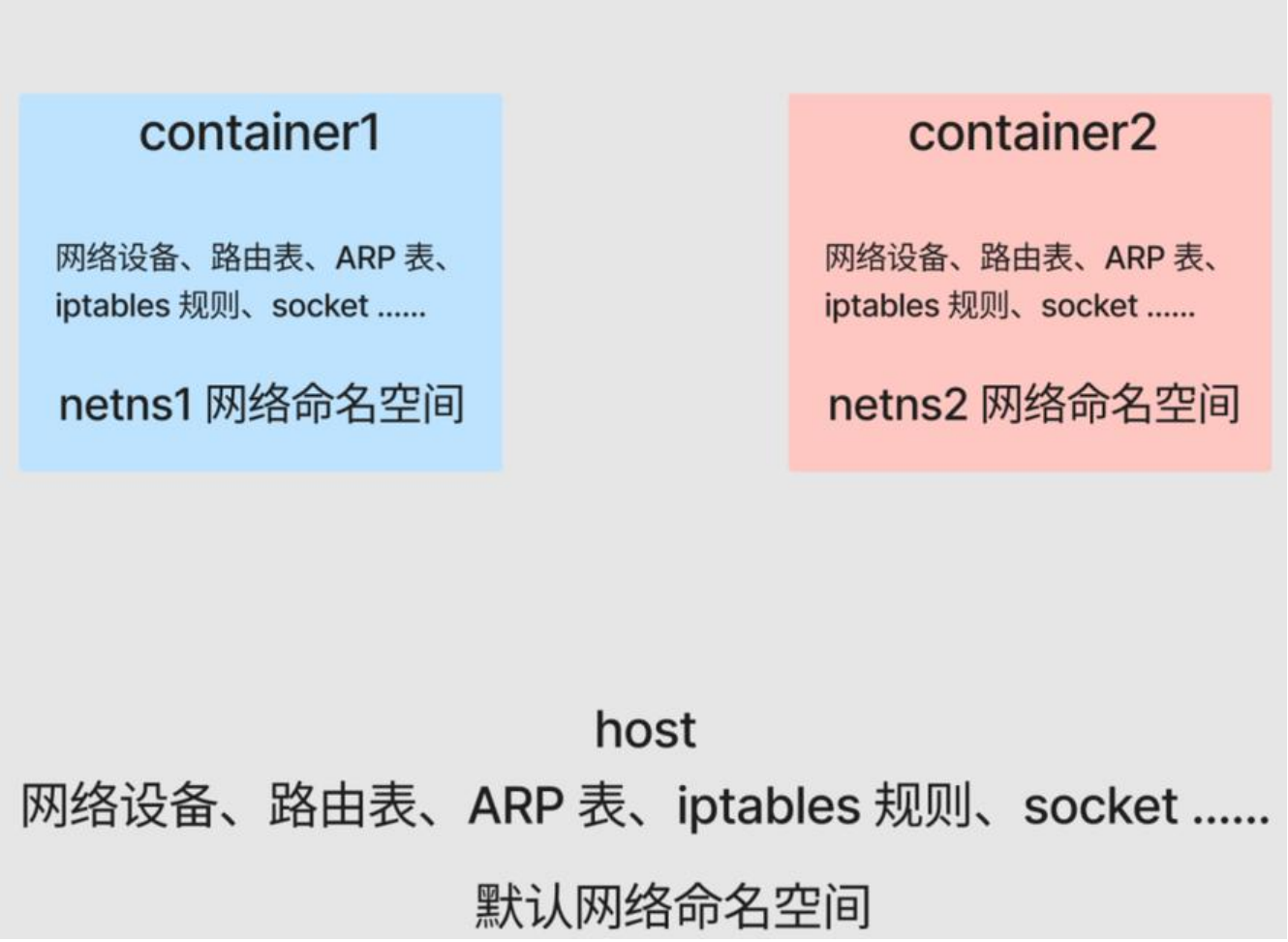
```
fangcong1> ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
fangcong1>
```

查看路由表

```
fangcong1> route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
fangcong1>
```

发现这个网络命名空间里的网络都被重置了，同样的fangcong2这个空间应该也如此。

不同的容器（ fangcong1 和 fangcong2 ）拥有自己独立的网络协议栈，包括网络设备、路由表、ARP 表、iptables 规则、socket 等，所有的容器都会以为自己运行在独立的网络环境中。

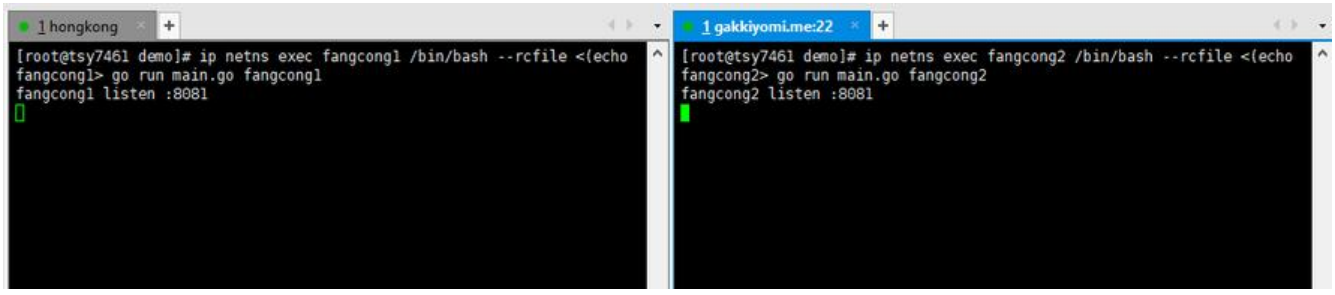


然后在测试一下 在fangcong1和fangcong2 的通信情况

准备一个go web服务

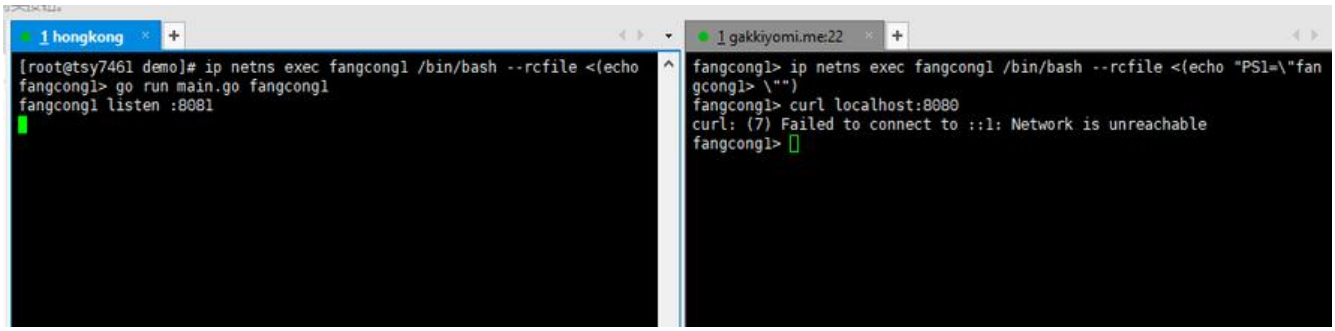
```
package main


import (
 "fmt"
 "net/http"
 "os"


)

func main() {
  name := os.Args[1]
  http.HandleFunc("/",func(w http.ResponseWriter, r *http.Request) {
   fmt.Println("req")
   w.Write([]byte(name + "\n"))


 })
 fmt.Println(name, "listen :8081")
 panic(http.ListenAndServe(":8081",nil))

}
```

然后在两个自定义的网络命名空间里运行

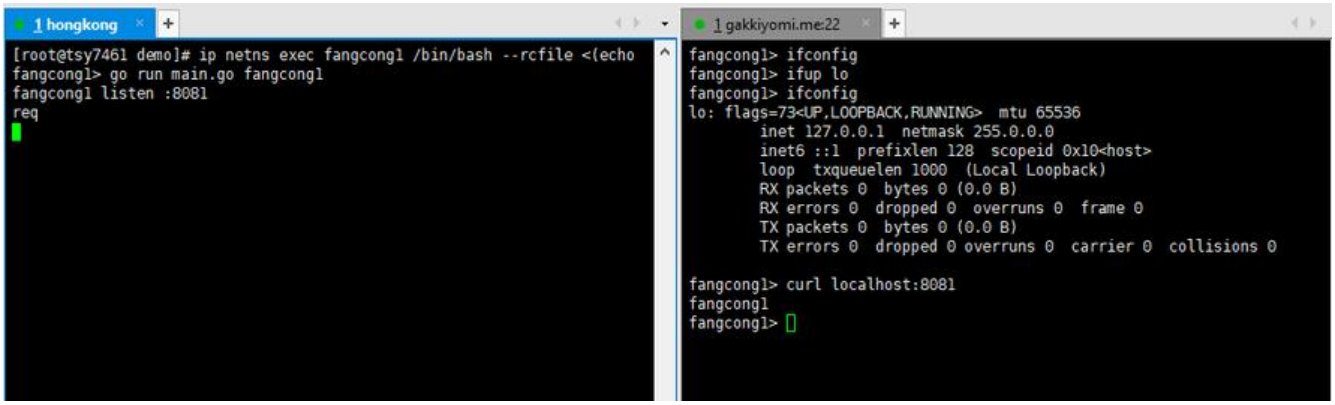


可以看到在同一主机下，起了两个8081端口却并没有发生冲突。

测试一下web服务器的可用性

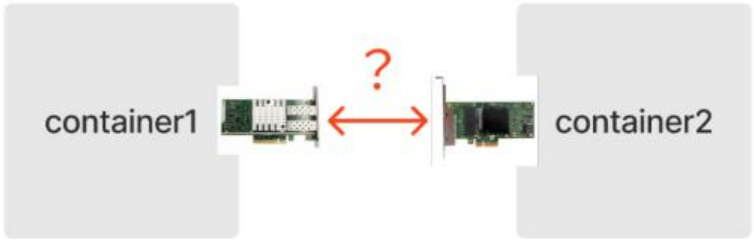


发现访问不通,因为还没有配置网卡

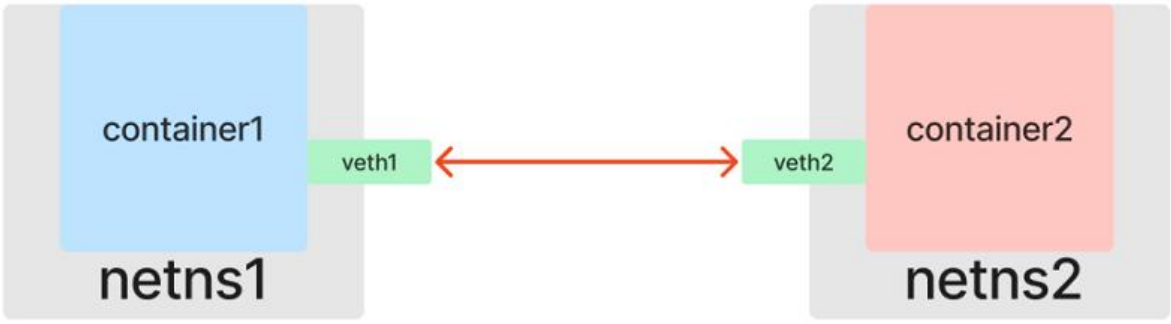


容器点对点通信

上面的小demo 证明当两个容器处于不同的Network Namespace中 他们的网络是隔离的,他们也无进行网络通信。在现实世界里如果两台计算机需要互通,只需要一根网线即可。那么容器呢?



在Linux网络虚拟化技术中提供了软件来模拟硬件网卡的方式,跟网线有两端一样，veth也是成对出现，被称为veth pair,只要将一对Veth分别放入两个Network Namespace中，这两个Network Namespace 就可以互相通信了。



创建veth pair

原文链接：从 Linux 网络虚拟化探究容器网络

```
[root@tsy7461 demo]# ip link add veth1 type veth peer name veth2
[root@tsy7461 demo]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 66:ba:f7:00:00:3a brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 02:42:40:5f:a9:f3 brd ff:ff:ff:ff:ff:ff
9: vethae9b639@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT gr
o default
    link/ether 9a:98:1c:e5:45:f1 brd ff:ff:ff:ff:ff:ff link-netnsid 2
23: veth06aaf88@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT
oup default
    link/ether 42:8f:59:e4:89:83 brd ff:ff:ff:ff:ff:ff link-netnsid 3
29: veth7d02136@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT
oup default
    link/ether ee:c9:df:a2:20:2d brd ff:ff:ff:ff:ff:ff link-netnsid 4
30: veth2@veth1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 36:26:9f:09:ad:29 brd ff:ff:ff:ff:ff:ff
31: veth1@veth2: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether aa:e9:7a:e7:cc:db brd ff:ff:ff:ff:ff:ff
[root@tsy7461 demo]#
```

将veth1 放入fangcong1，另一端veth2放入fangcong2

```
[root@tsy7461 demo]# ip link set veth1 netns fangcong1
[root@tsy7461 demo]# ip link set veth2 netns fangcong2
[root@tsy7461 demo]#
```

然后就可以在容器里看到对应的网络设备了

```
fangcong1> ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
31: veth1@if30: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether aa:e9:7a:e7:cc:db brd ff:ff:ff:ff:ff:ff link-netnsid 1
fangcong1>
```

然后分别为两个网卡设置ip地址，使其位于同一个子网 172.17.0.0/24然后启用网卡

```
fangcong1> ip addr add 172.17.0.101/24 dev veth1
fangcong1> ip link set dev veth1 up
fangcong1> ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10  bytes 941 (941.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 941 (941.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.101  netmask 255.255.255.0  broadcast 0.0.0.0
        ether aa:e9:7a:e7:cc:db  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

fangcong1>
```

```
fangcong2> ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.102  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::3426:9fff:fe09:ad29  prefixlen 64  scopeid 0x20<link>
        ether 36:26:9f:09:ad:29  txqueuelen 1000  (Ethernet)
        RX packets 8  bytes 656 (656.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 656 (656.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

fangcong2>
```

测试互访

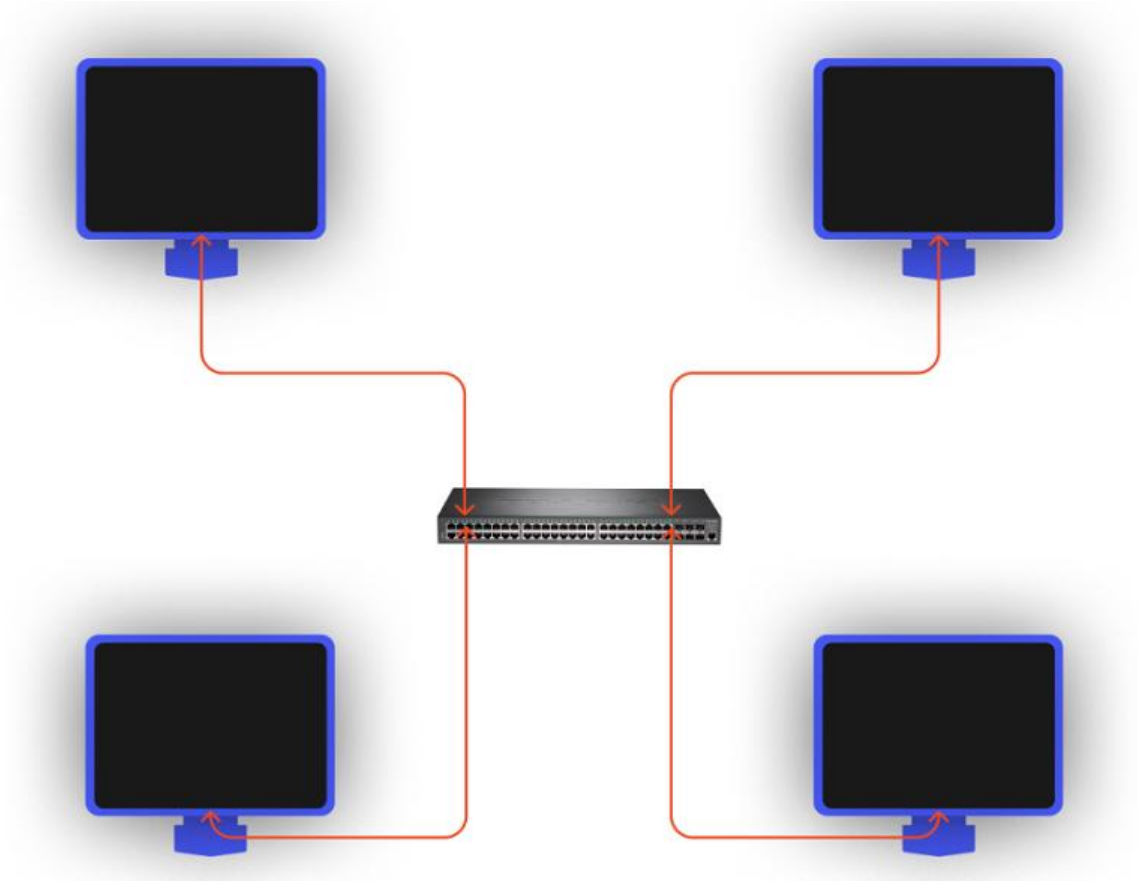
```
fangcong2> curl 172.17.0.101:8081
fangcong1
fangcong2>
```

```
fangcong1> go run main.go fangcong1
fangcong1 listen :8081
req
```

到这里容器点对点通信就成功解决了。

容器间互访

但是在真实的网络世界里 不可能只有两台计算机，肯定不是一个简单的二层网络，也没有足够多的网进行彼此之间的两两互联，所以就发明了二层交换机(或网桥)。



原文链接：从 Linux 网络虚拟化探究容器网络

容器也是如此，肯定会有3个或者以上的容器需要互访。那么就不能单单靠veth了。则需要用到linux我们提供的网桥虚拟化方式： Bridge。

先创建一个fangcong3的命名空间:

```
fangcong2> ip netns add fangcong3
fangcong2> ip netns list
fangcong3
fangcong2
fangcong1 (id: 0)
fangcong2>
```

然后将之前的veth1和veth2去除

```
fangcong1> ip link delete veth1
fangcong1> ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
fangcong1>
```

创建Bridge

```
[root@tsy7461 ~]#
[root@tsy7461 ~]# ip link add br0 type bridge
[root@tsy7461 ~]# ip link | grep br0
32: br0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
[root@tsy7461 ~]#
```

然后创建三对veth

```
[root@tsy7461 ~]# ip link add veth1 type veth peer name veth1-br
[root@tsy7461 ~]# ip link add veth2 type veth peer name veth2-br
[root@tsy7461 ~]# ip link add veth3 type veth peer name veth3-br
[root@tsy7461 ~]#
```

将 veth1 插入 fangcong1、veth1-br插入br0、veth2 插入fangcong2、veth2-br 插入 br0 、 veth3 插入 fangcong3、veth3-br插入br0 （记得启用 veth*-br ）：

```
[root@tsy7461 ~]#
[root@tsy7461 ~]# ip link add veth1 type veth peer name veth1-br
[root@tsy7461 ~]# ip link add veth2 type veth peer name veth2-br
[root@tsy7461 ~]# ip link add veth3 type veth peer name veth3-br
[root@tsy7461 ~]# ip link set dev veth1 netns fangcong1
[root@tsy7461 ~]# ip link set dev veth2 netns fangcong2
[root@tsy7461 ~]# ip link set dev veth3 netns fangcong3
[root@tsy7461 ~]# ip link set dev veth1-br master br0
[root@tsy7461 ~]# ip link set dev veth2-br master br0
[root@tsy7461 ~]# ip link set dev veth3-br master br0
[root@tsy7461 ~]# ip link set dev veth1-br up
[root@tsy7461 ~]# ip link set dev veth2-br up
[root@tsy7461 ~]# ip link set dev veth3-br up
[root@tsy7461 ~]#
```

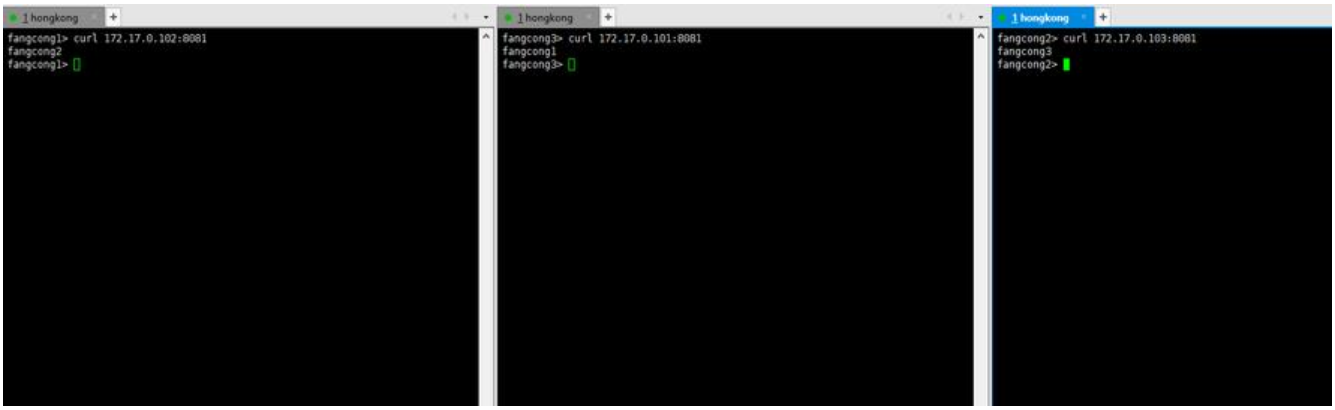
分别在三个容器中，为各自的网卡设置 IP 地址，并使其位于同一个子网 172.17.0.0/24 中，设置完同样需要进行启用操作：

```
fangcong1> ip addr add 172.17.0.101/24 dev veth1
fangcong1> ip link set dev veth1 up
fangcong1> ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
```

```
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10  bytes 941 (941.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 941 (941.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.101  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::cc78:8bff:fe0c:75ba  prefixlen 64  scopeid 0x20<link>
        ether ce:78:8b:0c:75:ba  txqueuelen 1000  (Ethernet)
        RX packets 6  bytes 516 (516.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 516 (516.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

fangcong2> ip addr add 172.17.0.102/24 dev veth2
fangcong2> ip lin set dev veth2 up
fangcong2> ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.102  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::64f5:52ff:fe95:3228  prefixlen 64  scopeid 0x20<link>
        ether 66:f5:52:95:32:28  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 266 (266.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 426 (426.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

fangcong3> ip addr add 172.17.0.103/24 dev veth3
fangcong3> ip link set dev veth3 up
fangcong3> ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.103  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::a405:23ff:fe99:60e2  prefixlen 64  scopeid 0x20<link>
        ether a6:05:23:99:60:e2  txqueuelen 1000  (Ethernet)
```

```
RX packets 6  bytes 516 (516.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 6  bytes 516 (516.0 B)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

测试三台容器互访

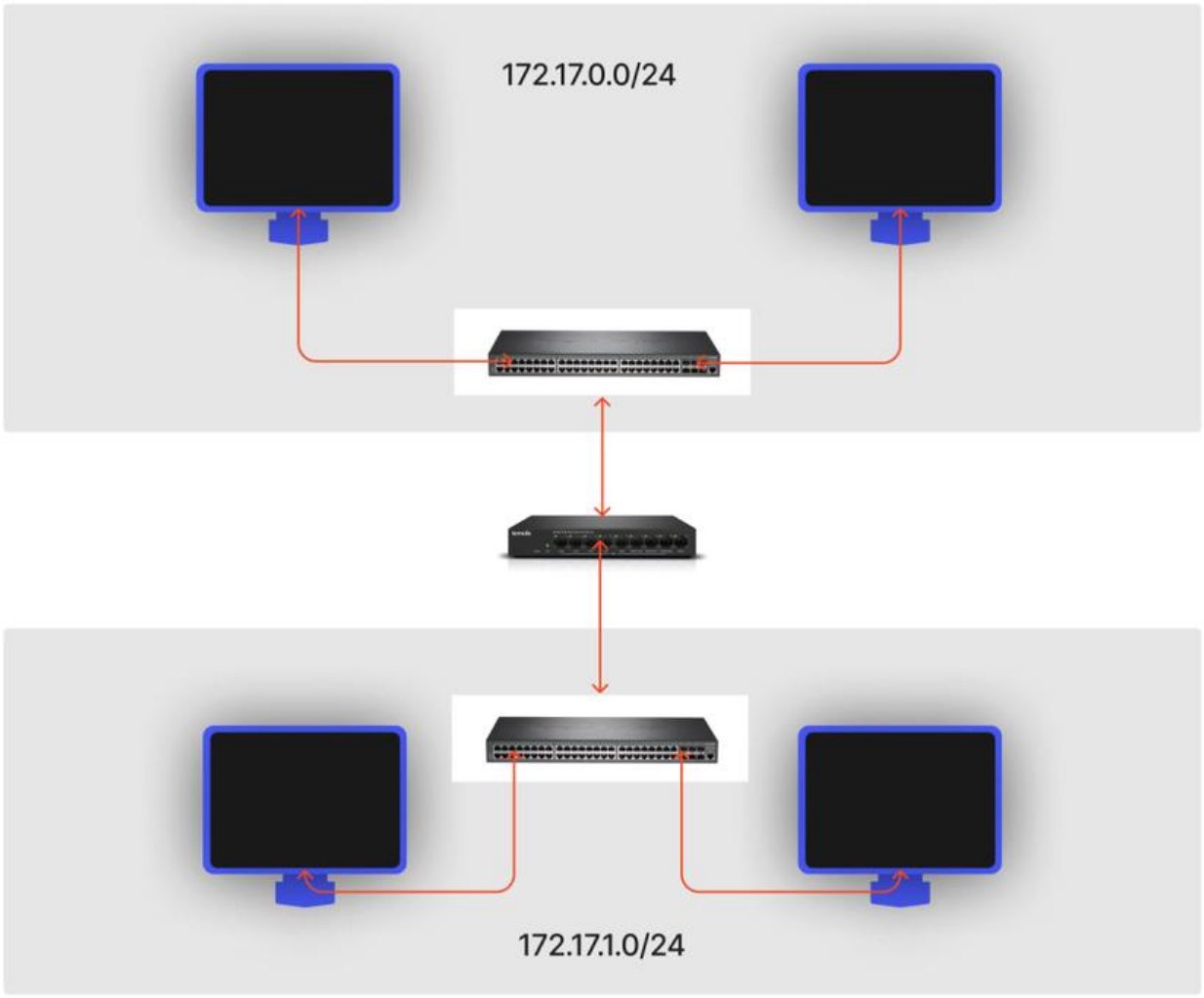


至此我们就实现了同宿主机上的多容器间互访，但现在距离docker还很远，所以还欠缺了以下几种网方案。

容器和宿主机互通

目前为止，我们的实验都是处于同一子网中。但实际的应用场景，更多的是需要容器可以与外部进行通。

在现实世界中，二层交换机只能解决同一子网内的数据流向，对于不同子网，就需要使用三层路由器或网关）来转发。



不过和之前 Linux 提供了交换机的虚拟化实现 Bridge 不同，Linux 并没有提供一个虚拟的路由器设。因为 Linux 其自身就已经具备了路由器的功能，可以直接用来充当路由器，更准确地说，在 Linux，一个 Network Namespace 就可以承担一个路由器的功能。

现在我们三个容器fangcong1,fangcong2,fangcong3三个容器处于同一子网172.17.0.0/24中，与宿机不在同一子网。宿主机的ip是103.239.101.157

```
[root@tsy7461 ~]# ifconfig
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::38ac:acff:fe79:69d  prefixlen 64  scopeid 0x20<link>
        ether 02:62:e2:e3:1e:05  txqueuelen 1000  (Ethernet)
        RX packets 33  bytes 1884 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 656 (656.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:40ff:fe5f:a9f3  prefixlen 64  scopeid 0x20<link>
        ether 02:42:40:5f:a9:f3  txqueuelen 0  (Ethernet)
        RX packets 9434657  bytes 5665217626 (5.2 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9237319  bytes 4559609837 (4.2 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 103.239.101.157  netmask 255.255.255.0  broadcast 103.239.101.255
        inet6 fe80::64ba:f7ff:fe00:3a  prefixlen 64  scopeid 0x20<link>
        ether 66:ba:f7:00:00:3a  txqueuelen 1000  (Ethernet)
        RX packets 10524971  bytes 1540008918 (1.4 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9260474  bytes 2826937833 (2.6 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

查看fangcong1容器的路由表

```
fangcong1> route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.17.0.0      0.0.0.0         255.255.255.0   U     0      0        0 veth1
fangcong1>
```

可以看到只有自己子网的路由规则。

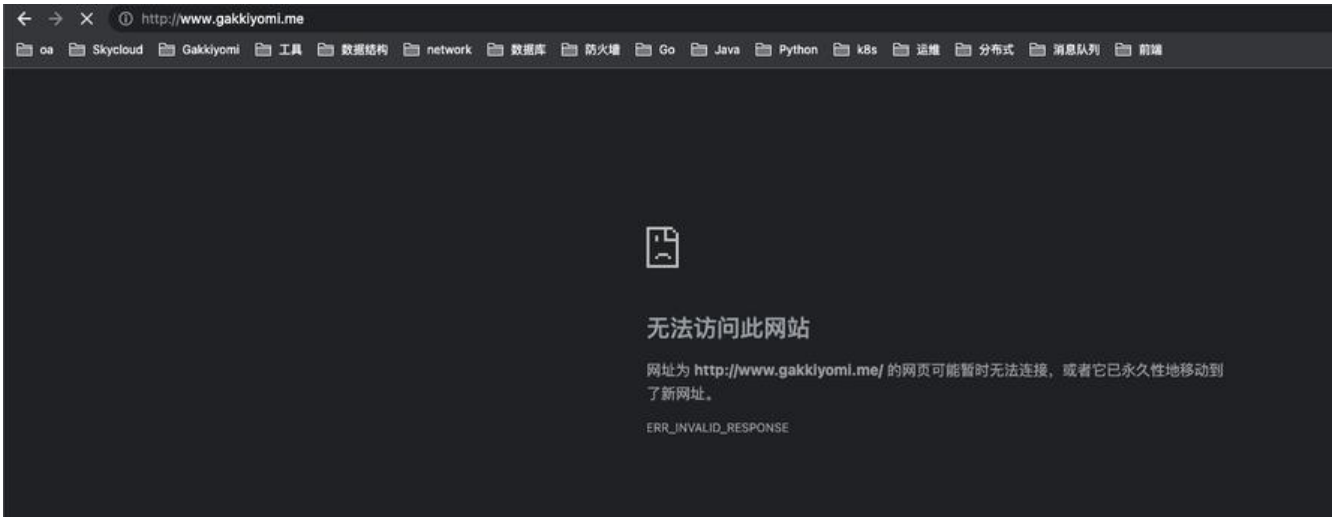
我们在宿主机上给网卡br0设置IP，让他充当三层网关来参与容器网络的路由转发寻路。（三台容器通过veth绑定到这个网卡上了）

```
[root@tsy7461 ~]# ip addr add local 172.17.0.1/24 dev br0
[root@tsy7461 ~]# ifconfig
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::38ac:acff:fe79:69d  prefixlen 64  scopeid 0x20<link>
        ether 02:62:e2:e3:1e:05  txqueuelen 1000  (Ethernet)
        RX packets 33  bytes 1884 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 656 (656.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

宿主机自动产生一条直连路由

```
[root@tsy7461 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         103.239.101.1   0.0.0.0         UG    100    0        0 eth0
103.239.101.0   0.0.0.0         255.255.255.0   U     100    0        0 eth0
172.17.0.0      0.0.0.0         255.255.255.0   U     0      0        0 br0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
```

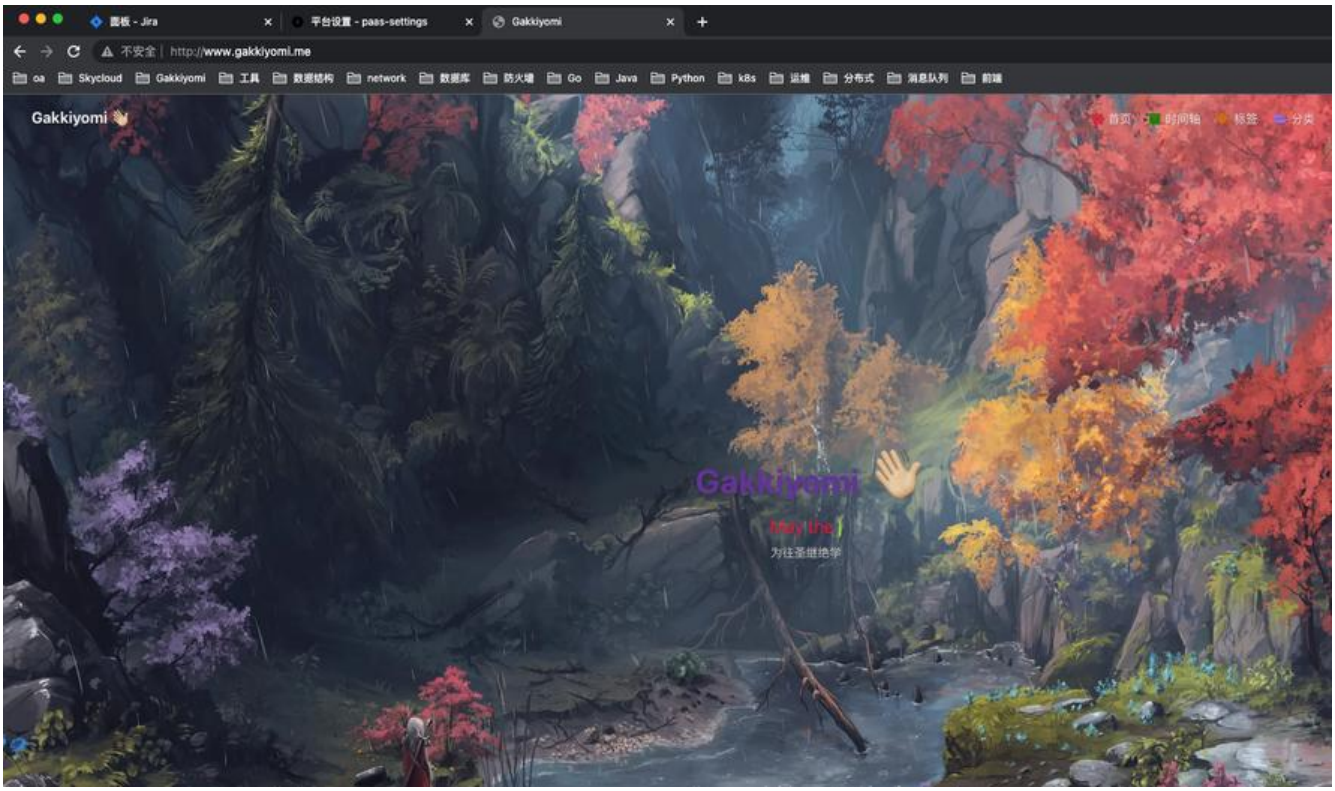
加上这条路由导致我的docker服务不通了,因为访问流量导入了br0的网卡，不走下面docker服务默的docker0的网桥了



删掉刚才配置的ip, 直连路由也自动消失，站点恢复。

```
[root@tsy7461 ~]# ip addr del local 172.17.0.1/24 dev br0
[root@tsy7461 ~]# ifconfig
```

```
[root@tsy7461 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         103.239.101.1   0.0.0.0         UG    100    0        0 eth0
103.239.101.0   0.0.0.0         255.255.255.0   U     100    0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
[root@tsy7461 ~]#
```



回到实验，我们加回接口ip，这时候通过新增的直连路由就可以实现宿主机访问容器了。

反过来，容器访问宿主机，也可以通过配置容器隔离的路由表来实现,这里我将br0的ip改成了172.17.02避免和docker0冲突

至此，我们通过三层路由表转发实现了容器与宿主机的网络通信

原文链接：从 Linux 网络虚拟化探究容器网络

容器访问其它主机（外网）

待续

外部访问容器（容器端口映射）

待续