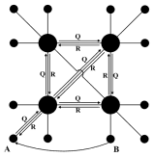


从Docker到Kubernetes 第5周

法律声明

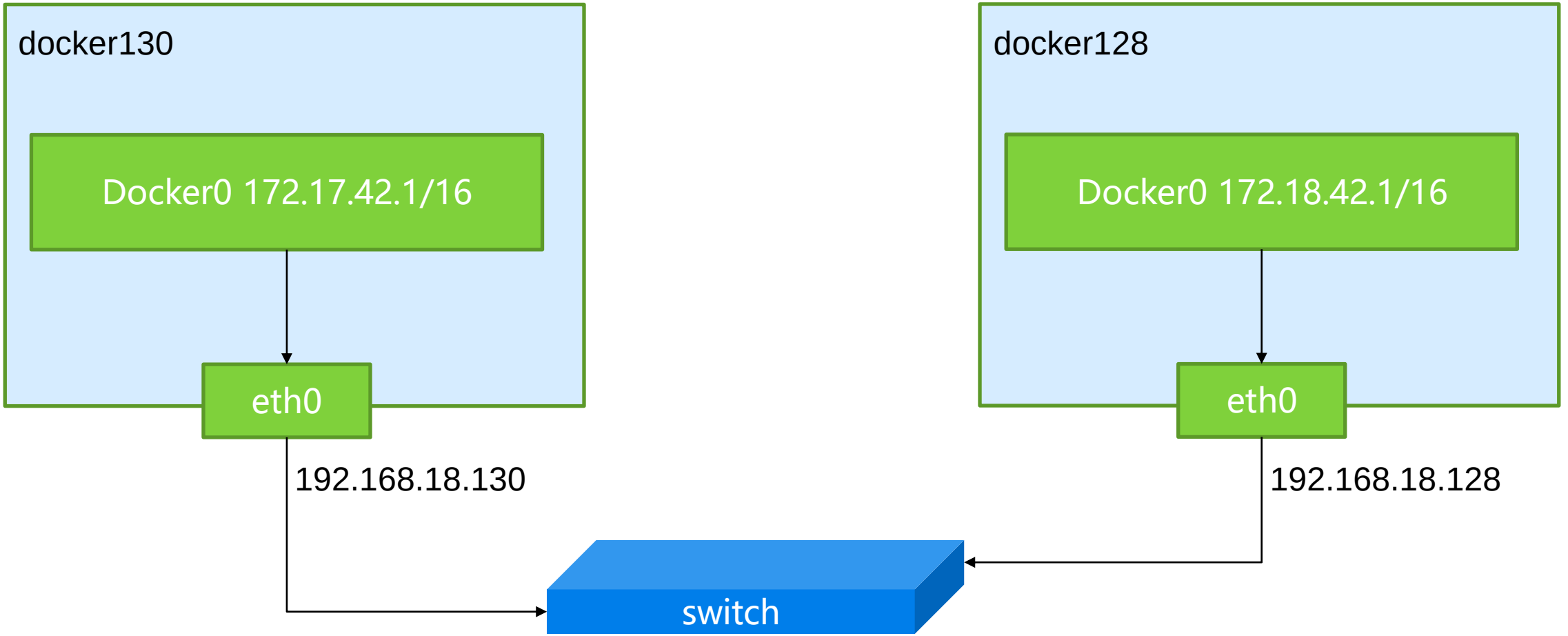
【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

- Linux路由机制打通网络
- Docker网络方案一览

Linux路由机制打通网络



Linux路由机制打通网络

docker128上修改Docker0的网络地址，与docker130不冲突

vi /usr/lib/systemd/system/docker.service

ExecStart=/usr/bin/docker daemon --bip=172.18.42.1/16 -H fd:// -H=unix:///var/run/docker.sock

systemctl daemon-reload

重启docker128

```
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:e8:02:c7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.18.128/24 brd 192.168.18.255 scope global dynamic eth0
       valid_lft 1738sec preferred_lft 1738sec
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:43:93:3e:0c brd ff:ff:ff:ff:ff:ff
    inet 172.18.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
```

Linux路由机制打通网络

docker130 上执行 route add -net 172.18.0.0/16 gw 192.168.18.128
docker128 上执行 route add -net 172.17.0.0/16 gw 192.168.18.130

```
[root@docker128 ~]# ip route
default via 192.168.18.2 dev eth0  proto static  metric 100
172.17.0.0/16 via 192.168.18.130 dev eth0
172.18.0.0/16 dev docker0  proto kernel  scope link  src 172.18.42.1
192.168.18.0/24 dev eth0  proto kernel  scope link  src 192.168.18.128  metric 100
```

130上启动一个容器，获取其IP地址

```
[root@docker130 ~]# docker run --rm=true -it java /bin/bash
root@da98a79c83af:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:01 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global eth0
       valid_lft forever preferred_lft forever
```

128上Ping容器

```
[root@docker128 ~]# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
From 192.168.18.130 icmp_seq=1 Destination Host Prohibited
From 192.168.18.130 icmp_seq=2 Destination Host Prohibited
From 192.168.18.130 icmp_seq=3 Destination Host Prohibited
```

Linux路由机制打通网络

防火墙规则导致Ping禁止

```
[root@docker130 ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere             state NEW tcp dpt:ssh
REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited
```

iptables -F ; iptables -t nat -F

```
[root@docker128 ~]# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=63 time=0.960 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=63 time=1.19 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=63 time=0.723 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=63 time=0.891 ms
```

```
[root@docker128 ~]#  docker run --rm=true -it java /bin/bash
root@78e10ba9768f:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state U
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noq
    link/ether 02:42:ac:12:00:01 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 scope global eth0
       valid_lft forever preferred_lft forever
root@78e10ba9768f:/# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1): 56 data bytes
64 bytes from 172.17.0.1: icmp_seq=0 ttl=62 time=1.182 ms
64 bytes from 172.17.0.1: icmp_seq=1 ttl=62 time=1.427 ms
```

Linux路由机制打通网络

Docker130上的一个容器ping 128上的一个容器

```
root@da98a79c83af:/# ping 172.18.0.1
PING 172.18.0.1 (172.18.0.1): 56 data bytes
64 bytes from 172.18.0.1: icmp_seq=0 ttl=62 time=1.337 ms
64 bytes from 172.18.0.1: icmp_seq=1 ttl=62 time=1.206 ms
64 bytes from 172.18.0.1: icmp_seq=2 ttl=62 time=1.304 ms
64 bytes from 172.18.0.1: icmp_seq=3 ttl=62 time=1.427 ms
```

Docker128上抓包看到结果

```
[root@docker130 ~]# tshark -f icmp
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
  1    0.000000    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0009, seq=0/0, ttl=63
  2    0.000821    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0009, seq=0/0, ttl=63 (request in 1)
2   3    1.001374    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0009, seq=1/256, ttl=63
  4    1.002220    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0009, seq=1/256, ttl=63 (request in 3)
```

```
[root@docker130 ~]# tshark -i docker0 -f icmp
Running as user "root" and group "root". This could be dangerous.
Capturing on 'docker0'
  1    0.000000    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0009, seq=0/0, ttl=64
  2    0.001050    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0009, seq=0/0, ttl=62 (request in 1)
2   3    1.001468    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0009, seq=1/256, ttl=64
  4    1.002446    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0009, seq=1/256, ttl=62 (request in 3)
4   5    2.003515    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0009, seq=2/512, ttl=64
```

Linux路由机制打通网络

Docker130上的容器 c1:172.17.0.1 ping 128上的容器c2:172.18.0.1时，c1发现这个地址不是自己子网的，于是发给docker0网关

```
Capturing on 'docker0'
  1    0.000000    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x000f, seq=0/0, ttl=64
```

经过路由计算，这个报文被发往下一跳的路由器端口:eth0，所以ttl减一

```
Capturing on 'eth0'
  1    0.000000    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x000f, seq=0/0, ttl=63
```

报文到达128主机的eth0网卡，经过路由计算，被发往下一跳的端口dock0：

```
Capturing on 'eth0'
  1    0.000000    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0010, seq=0/0, ttl=63
```

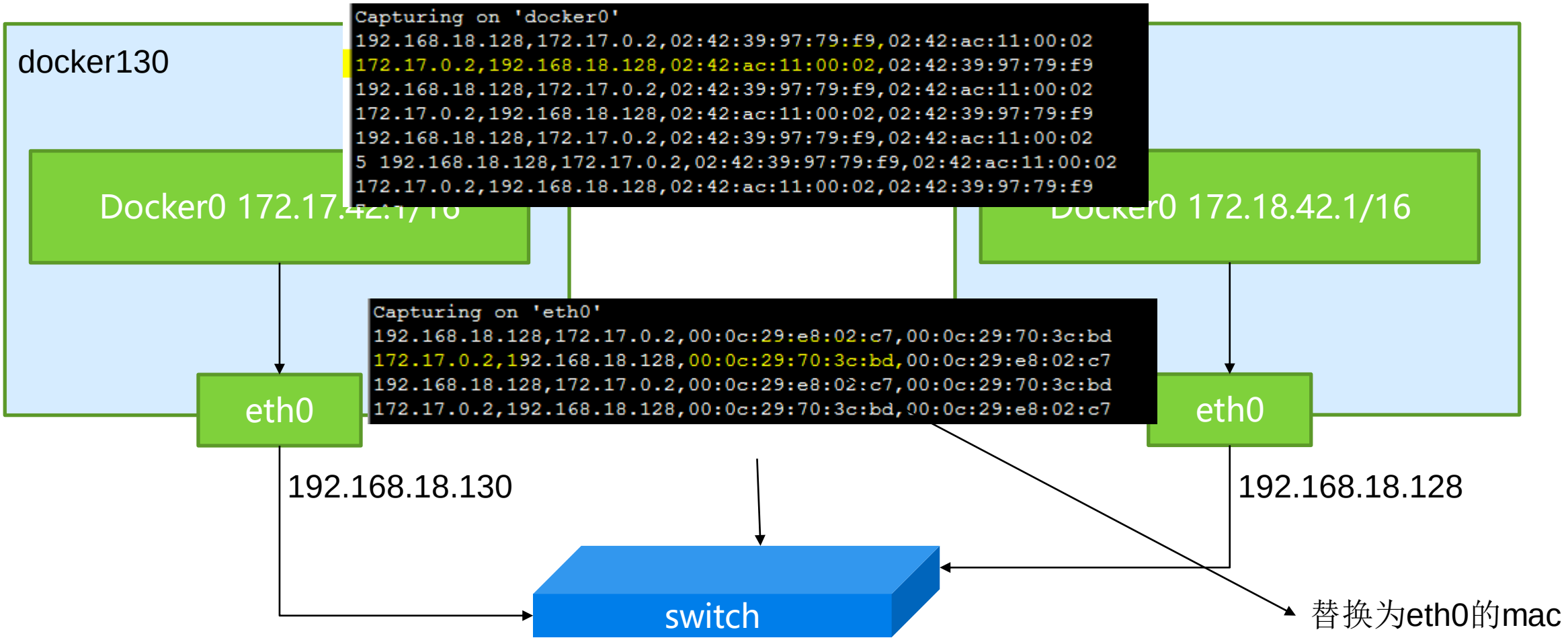
注意到docker0上的ttl又减了一

```
Capturing on 'docker0'
  1    0.000000    172.17.0.1 -> 172.18.0.1    ICMP 98 Echo (ping) request  id=0x0010, seq=0/0, ttl=62
```

回来的时候，数据包流程:c2→128 docker0 → 128 eth0 → 130 eth0 → 130 docker0 →c1

```
  2    0.000110    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0010, seq=0/0, ttl=64 (request in 1)
  2    0.000230    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0010, seq=0/0, ttl=63 (request in 1)
  2    0.000230    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x0010, seq=0/0, ttl=63 (request in 1)
  2    0.000724    172.18.0.1 -> 172.17.0.1    ICMP 98 Echo (ping) reply    id=0x000f, seq=0/0, ttl=62 (request in 1)
```

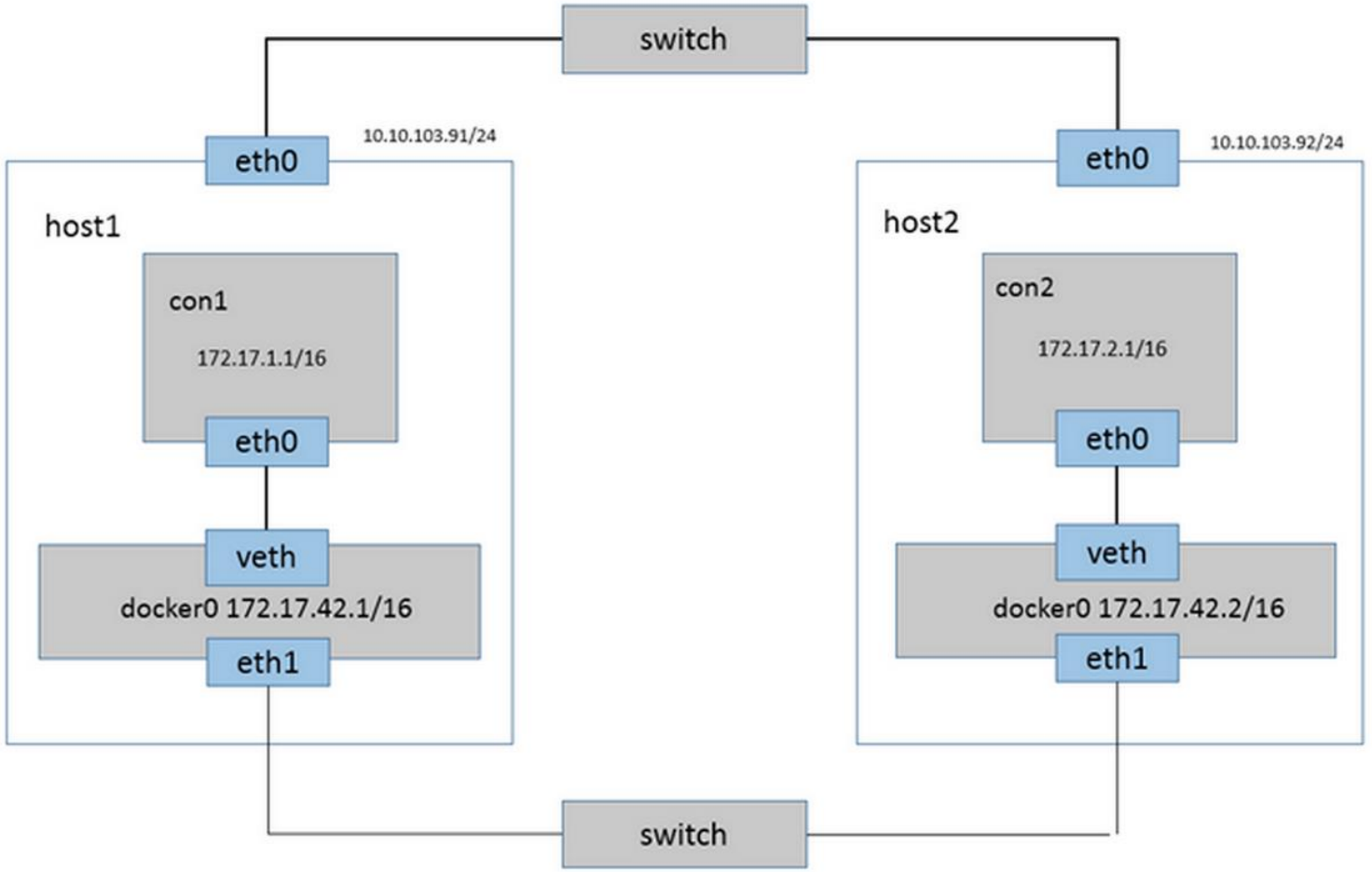
Linux路由机制打通网络



tshark -N m -e ip.src -e ip.dst -e eth.src -e eth.dst -Tfields -E separator=, -f "net 172.17.0.0/16"

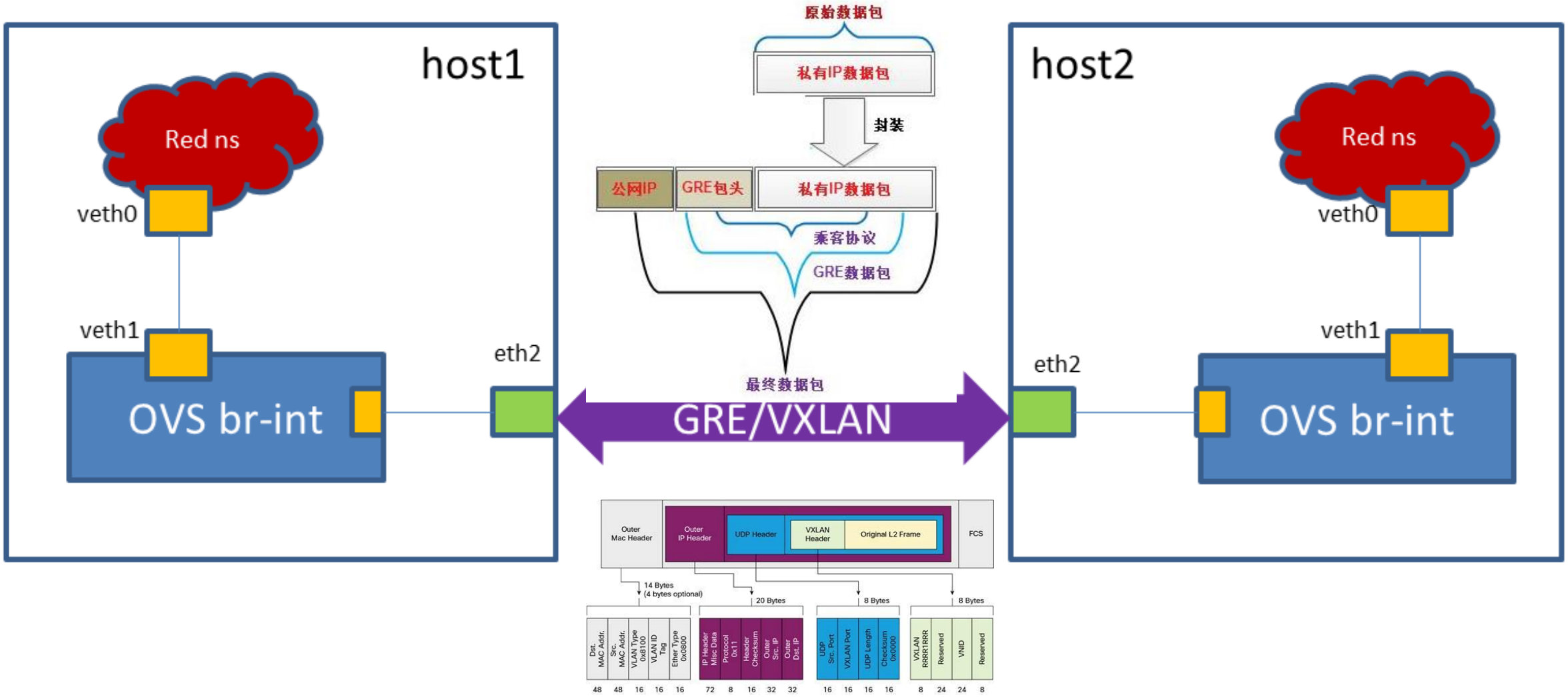
Docker网络方案一览

双网卡独立大二层交换（linux bridge)



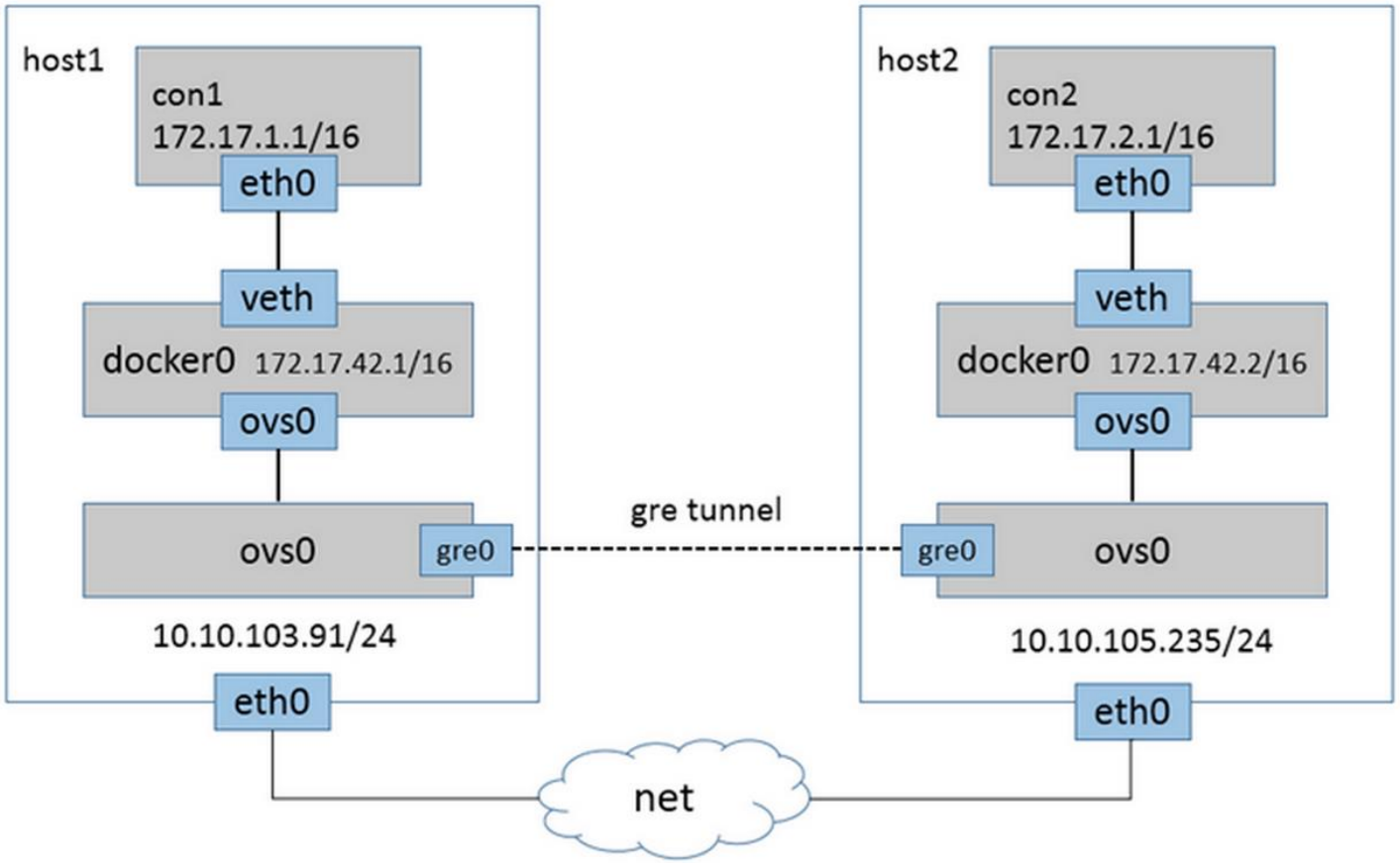
Docker网络方案一览

Overlay网络



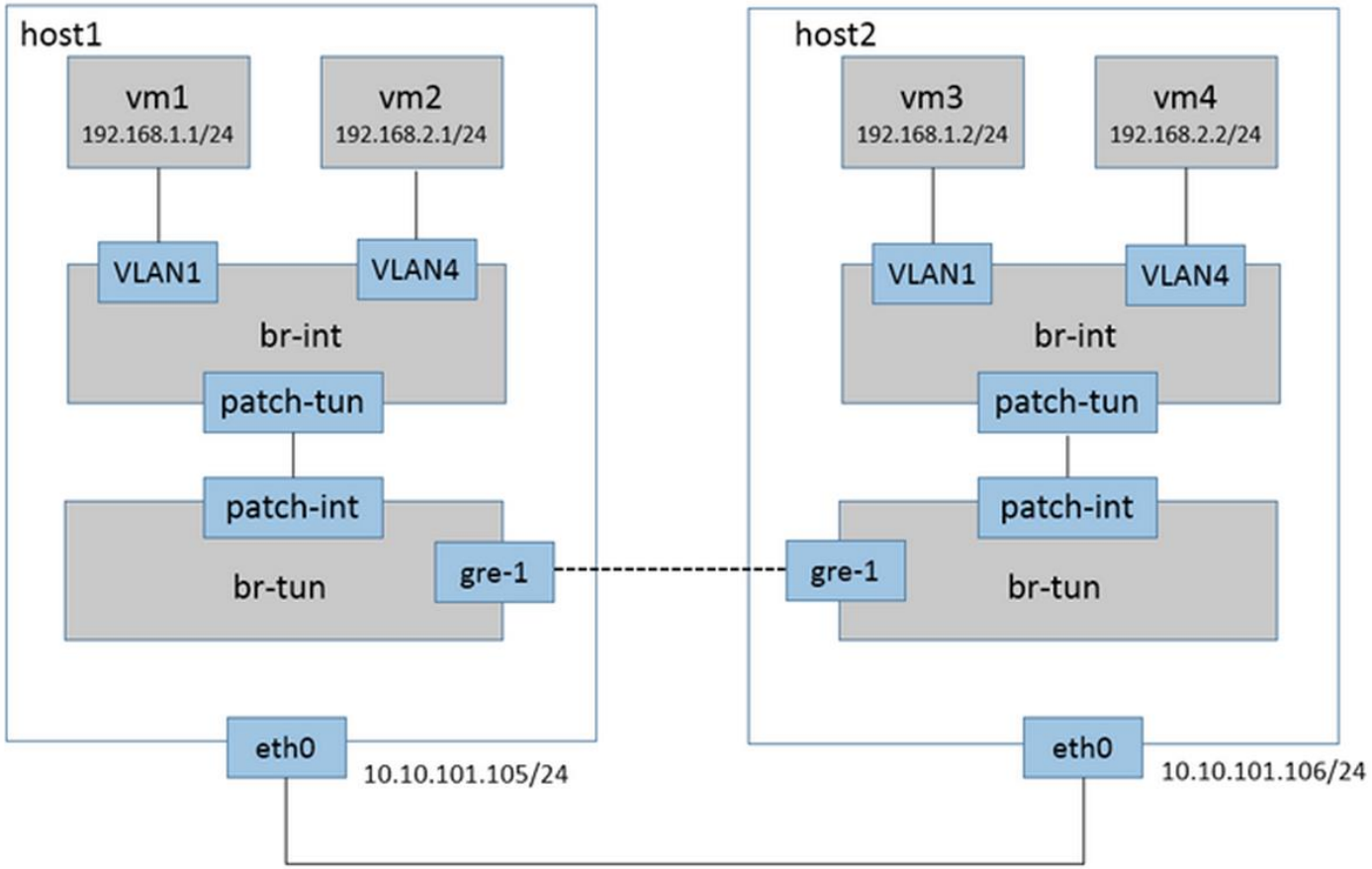
Docker网络方案一览

基于ovs的Overlay网络



Docker网络方案一览

学习neutron网络



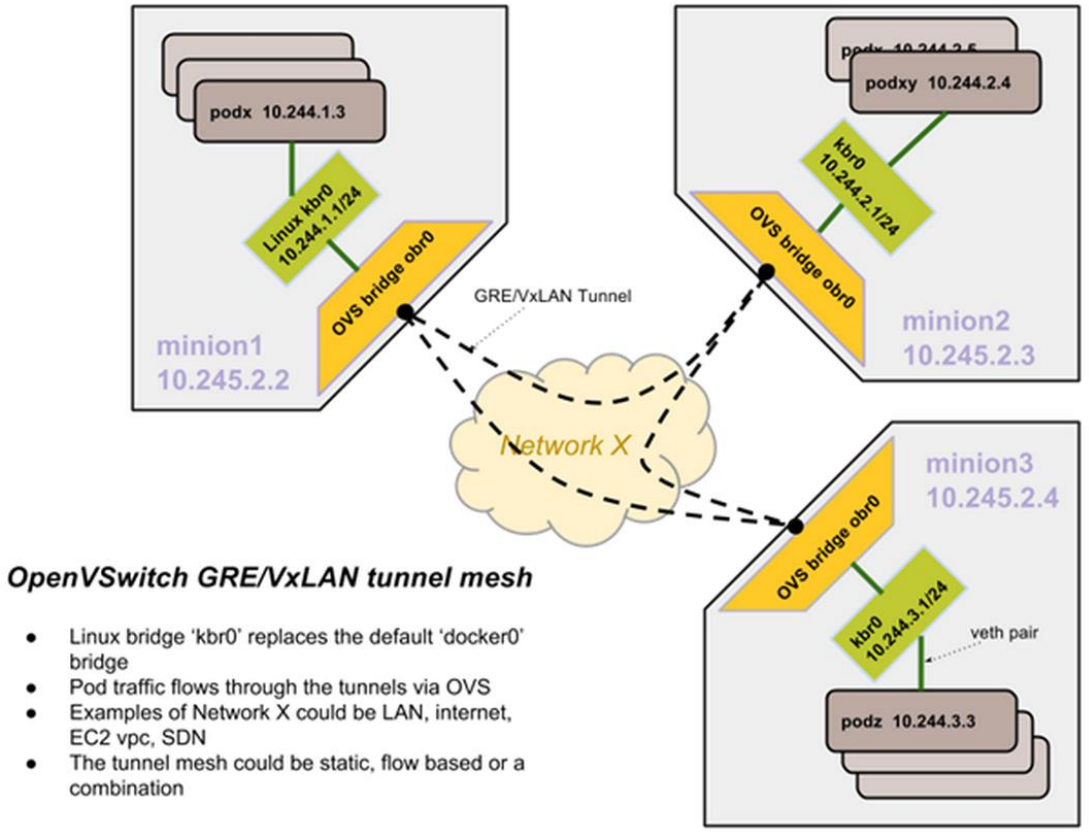
Docker网络方案一览

官方的Libnetwork

Socketplane被docker公司收购，成为其官方网络的起源

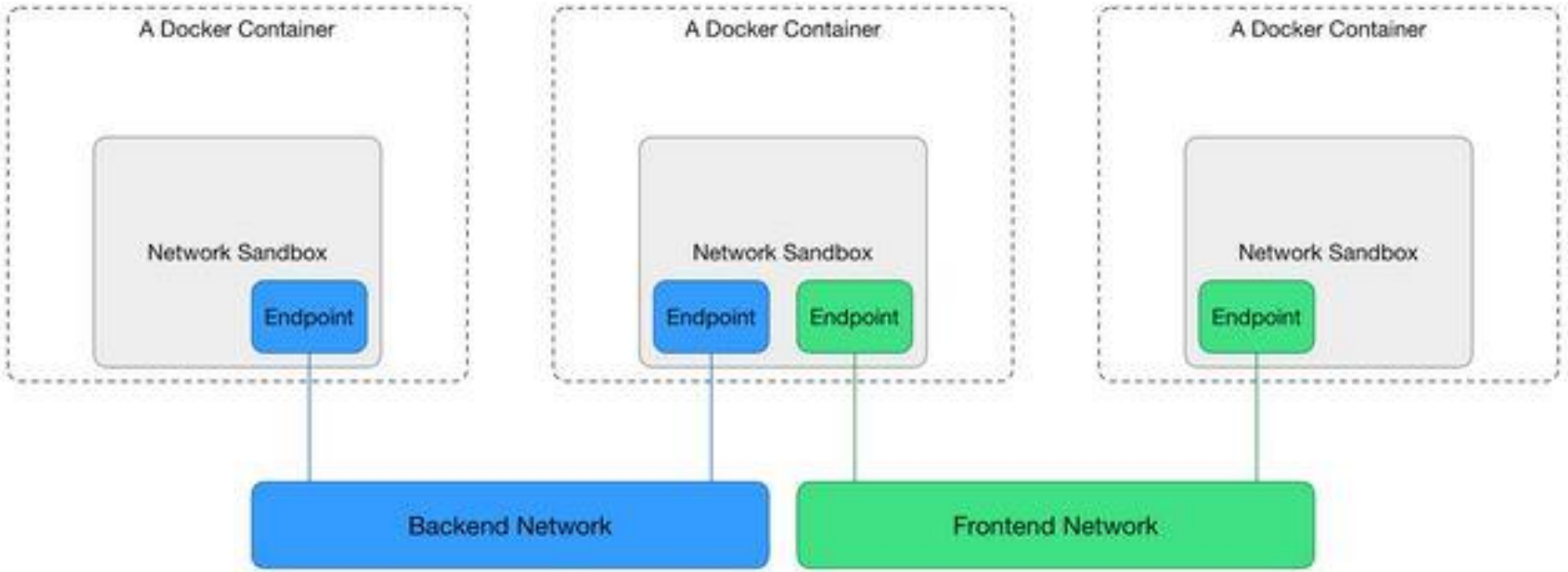
开发者不想操作是否是 VLANs, VXLANs, Tunnels 或者是 TEPs. 对于架构人们最关心的是性能和可靠性。而 SocketPlane 在 socket 层面提供了一个网络的抽象层，通过可管理的方式去解决各种网络问题。

主要特性：
•Open vSwitch 集成
•用于 Docker 的零配置多主机网络
•Docker/SocketPlane 集群的优雅增长
•支持多网络
•分布式 IP 地址管理 (IPAM)



Docker网络方案一览

官方的Libnetwork



1. 将libnetwork集成到Docker Engine
2. 在Docker CLI中使用新的network命令
3. 撰写『-net』参数的文档，以告知用户如何使用它来为容器指定网络
4. 在network和endpoint中支持添加『label』
5. 研发新的『bridge』插件，以替换Docker目前的实现
6. 研发『分布式bridge』插件，以支持跨容器网络

Docker网络方案一览

官方的Libnetwork

目前看到有一些计划是打算将OVS项目关联到Docker上来，从Linux Kernel 3.3开始，OVS项目就是内核的一部分。当我听到这个的时候我觉得是不是脑袋让驴踢了。首先声明我并不是反对使用OVS，实际上，它是一个非常不错的网络工具套件。它的设置比较复杂，对于新手来说有一个陡峭的学习曲线，但是一旦学会，OVS就可以帮你事半功倍。关于这个话题我听到的一个讨论是：“如果OVS工作在Docker上，那么工作一切都变得很美好”。让我告诉你，亲们：如果让我花费大量时间学习它，最后的结果只能是：“还好，可以用”。我并不想说的那么愤世嫉俗，实际情况是在某些常用环境下OVS会崩溃。因此，使用OVS只是一种疯狂的想法罢了。

http://containertutorials.com/network/ovs_docker.html

https://github.com/openvswitch/ovs/blob/master/utilities/ovs-docker

Docker网络方案一览

官方的Libnetwork

Currently libnetwork is nothing more than an attempt to modularize the Docker platform's networking subsystem by moving it into libnetwork as a library.

这仅仅是开始

Thanks

FAQ时间