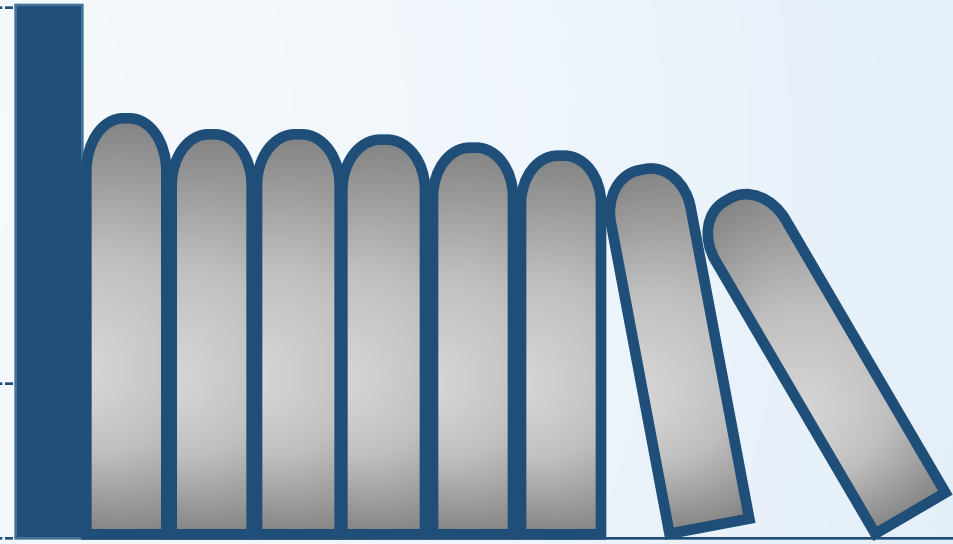
Docker网络模式

主讲人：宋小金



目录

1 预期收获

- 了解容器内外通信技术
- 了解容器跨主机通信
- 网络模式选型
- 通过实践加深理解

Docker网络模式

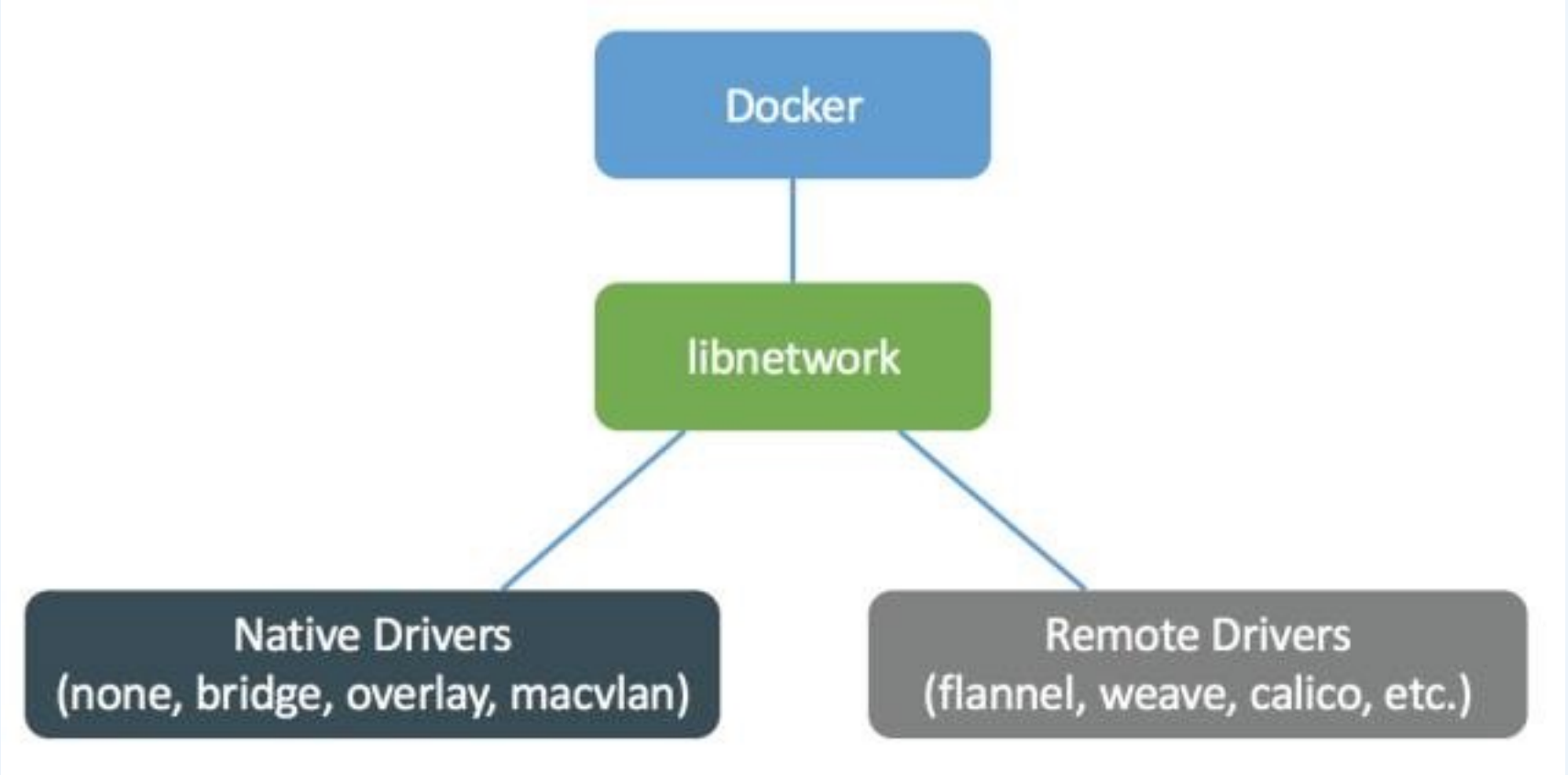
先介绍两个重要概念：

- veth设备全称为Virtual Enternet device，veth主要的目的是为了*跨Network Namespace之间*提供一种类似于Linux进程间通信的技术，所以veth总是成对出现的，例如veth0@veth1等，其中*veth0在一个Network Namespace内，而另一个veth1在另外一个Network Namespace*，其中往veth设备上任意一端上RX到的数据，都会在另一端上以TX的方式发送出去，veth工作在*L2数据链路层*，veth-pair设备在转发数据包过程中并不串改数据包内容。

- 网络命名空间是用于*隔离网络资源*（/proc/net、IP 地址、网卡、路由等）。由于一个物理的网络设备最多存放在一个网络命名空间中，所以通过 *veth pair 在不同的网络命名空间中创建通道*，才能达到通信的目的。

Docker网络模式

网络图示



Docker网络模式

先介绍两个重要概念：

- veth设备全称为Virtual Enternet device，veth主要的目的是为了*跨Network Namespace之间*提供一种类似于Linux进程间通信的技术，所以veth总是成对出现的，例如veth0@veth1等，其中*veth0在一个Network Namespace内，而另一个veth1在另外一个Network Namespace*，其中往veth设备上任意一端上RX到的数据，都会在另一端上以TX的方式发送出去，veth工作在*L2数据链路层*，veth-pair设备在转发数据包过程中并不串改数据包内容。

- 网络命名空间是用于*隔离网络资源*（/proc/net、IP 地址、网卡、路由等）。由于一个物理的网络设备最多存放在一个网络命名空间中，所以通过 *veth pair 在不同的网络命名空间中创建通道*，才能达到通信的目的。

Docker网络模式

Linux下的Docker容器网络是通过***Network Namespace机制实现隔离***的，不同的Network Namespace有各自的*网络设备、协议栈、路由表以及防火墙*等，同一个Namespace下的进程共享同一个网络视图。通过 *docker run* 启动的一个容器便具有了单独的网络命名空间，Docker提供了5种网络模式:

- Bridge模式
- Host模式
- Container模式
- None
- overlay

Docker网络模式

查看Docker Network:

```
!5012 $ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0701c965e832        bridge              bridge              local
733578948fb5        host                host                local
b34700b13beb        none                null                local
3f7a148e53d8        splunk              bridge              local
```

Docker网络模式 - Bridge 模式

Bridge 模式

- Bridge模式是Docker默认的一种网络通信模式，Docker Daemon第一次启动时，会在其所在的宿主机上创一个名叫***docker0的虚拟网桥***。Docker利用veth pair技术，在一个容器启动时，会创建一对虚拟网络接口veth pair，Docker会将***一端挂载到虚拟网桥docker0***上，而将veth pair的***另一端放在相关容器的Network Namespace***内。

- 在Bridge模式下是通过***iptables控制容器与Internet通信、以及容器间通信***的。docker0网桥会为每一个容器分配一个新的 IP 地址，并将 docker0 的 IP 地址设置为默认的网关。网桥 docker0 ***通过 iptables 中的配置***与宿主机器上的网卡相连，所有符合条件的请求都会***通过 iptables 转发到 docker0*** 并由网桥分发给对应的机器。

Docker网络模式 - Bridge 模式

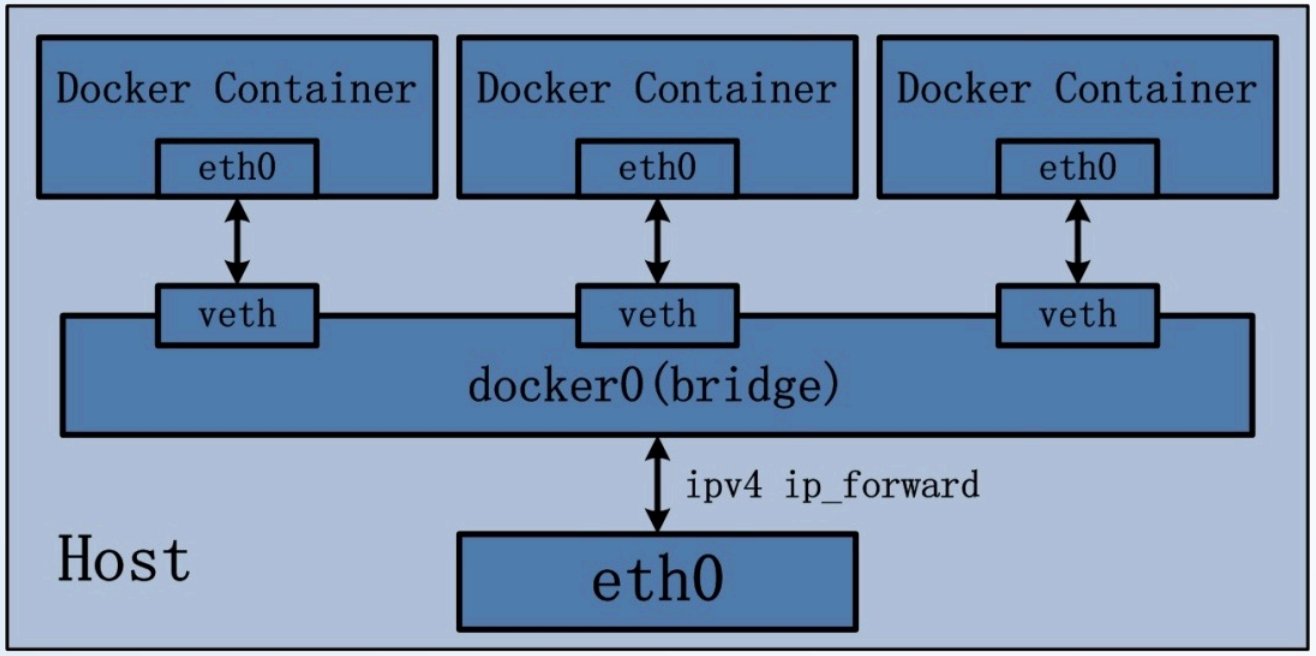
Bridge 模式

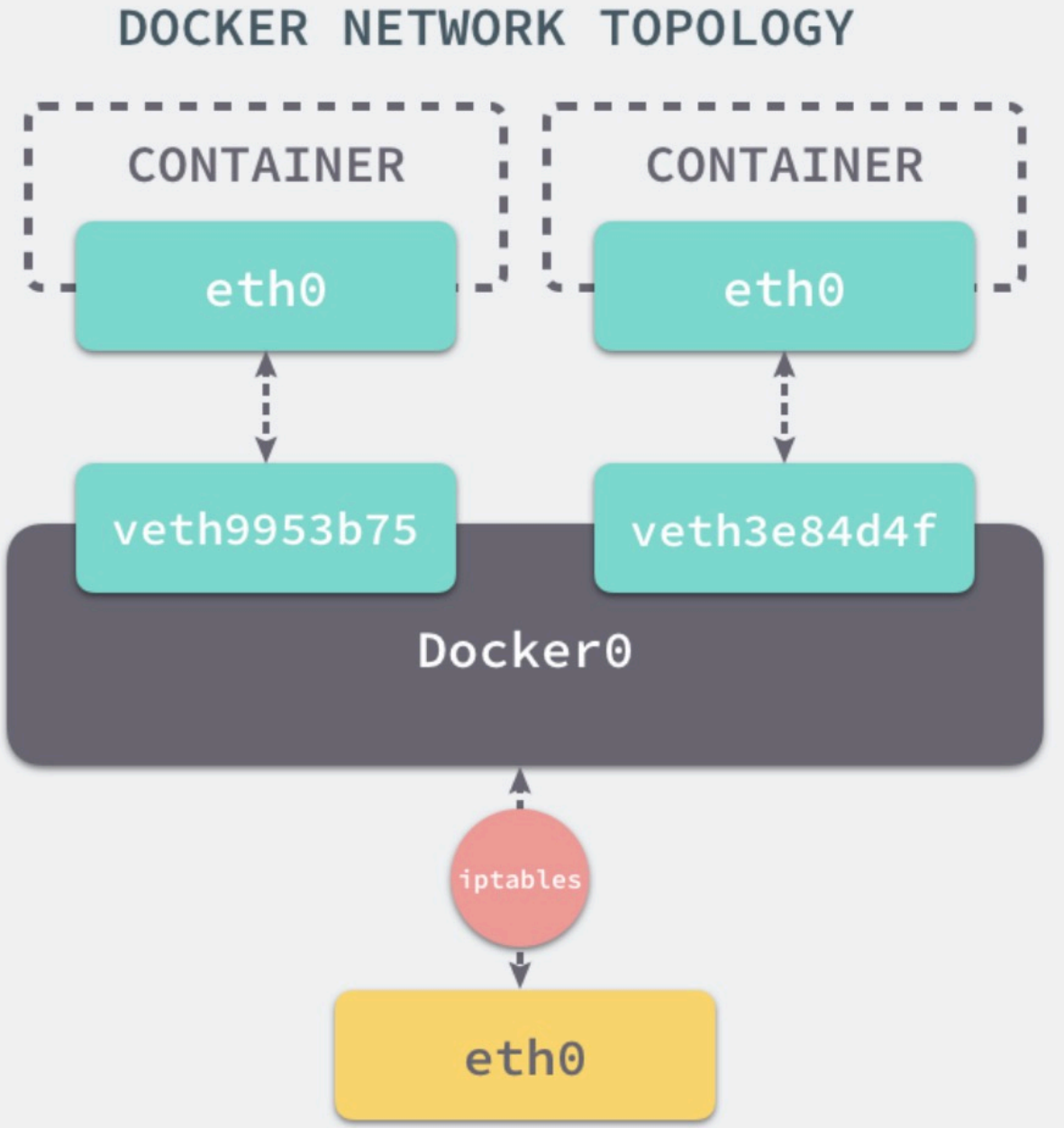
- Bridge模式是Docker默认的一种网络通信模式，Docker Daemon第一次启动时，会在其所在的宿主机上创一个名叫***docker0的虚拟网桥***。Docker利用veth pair技术，在一个容器启动时，会创建一对虚拟网络接口veth pair，Docker会将***一端挂载到虚拟网桥docker0***上，而将veth pair的***另一端放在相关容器的Network Namespace***内。

- 在Bridge模式下是通过***iptables控制容器与Internet通信、以及容器间通信***的。docker0网桥会为每一个容器分配一个新的 IP 地址，并将 docker0 的 IP 地址设置为默认的网关。网桥 docker0 ***通过 iptables 中的配置***与宿主机器上的网卡相连，所有符合条件的请求都会***通过 iptables 转发到 docker0*** 并由网桥分发给对应的机器。

Docker网络模式 - Bridge 模式

Bridge模式示意图





Docker网络模式 - Bridge 模式

启动Docker容器前后， 查看iptables规则的变化

请参考这里

对比启动Nginx容器之后， 当前 iptables的NAT 配置在 DOCKER 链中出现了一条新规则：

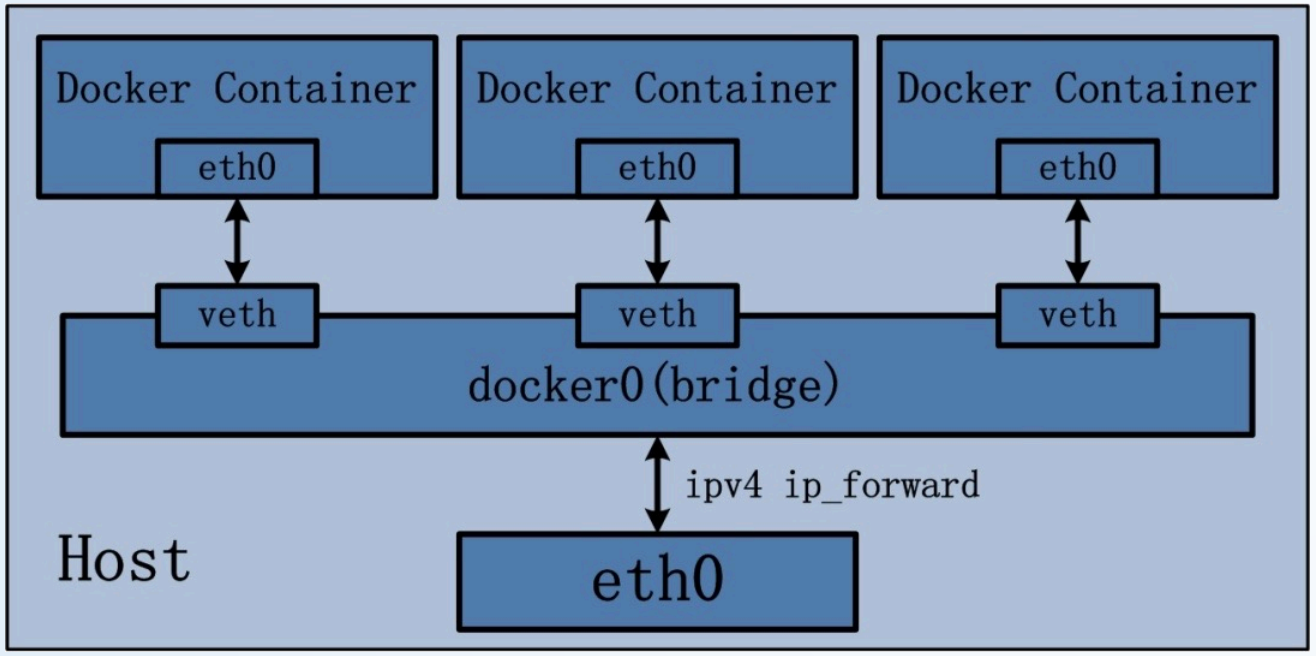
```
16a17
> MASQUERADE  tcp  --  172.17.0.2       172.17.0.2       tcp dpt:6379
22a24
> DNAT       tcp  --  anywhere         anywhere         tcp dpt:6379 to:172.17.0.2:6379
```

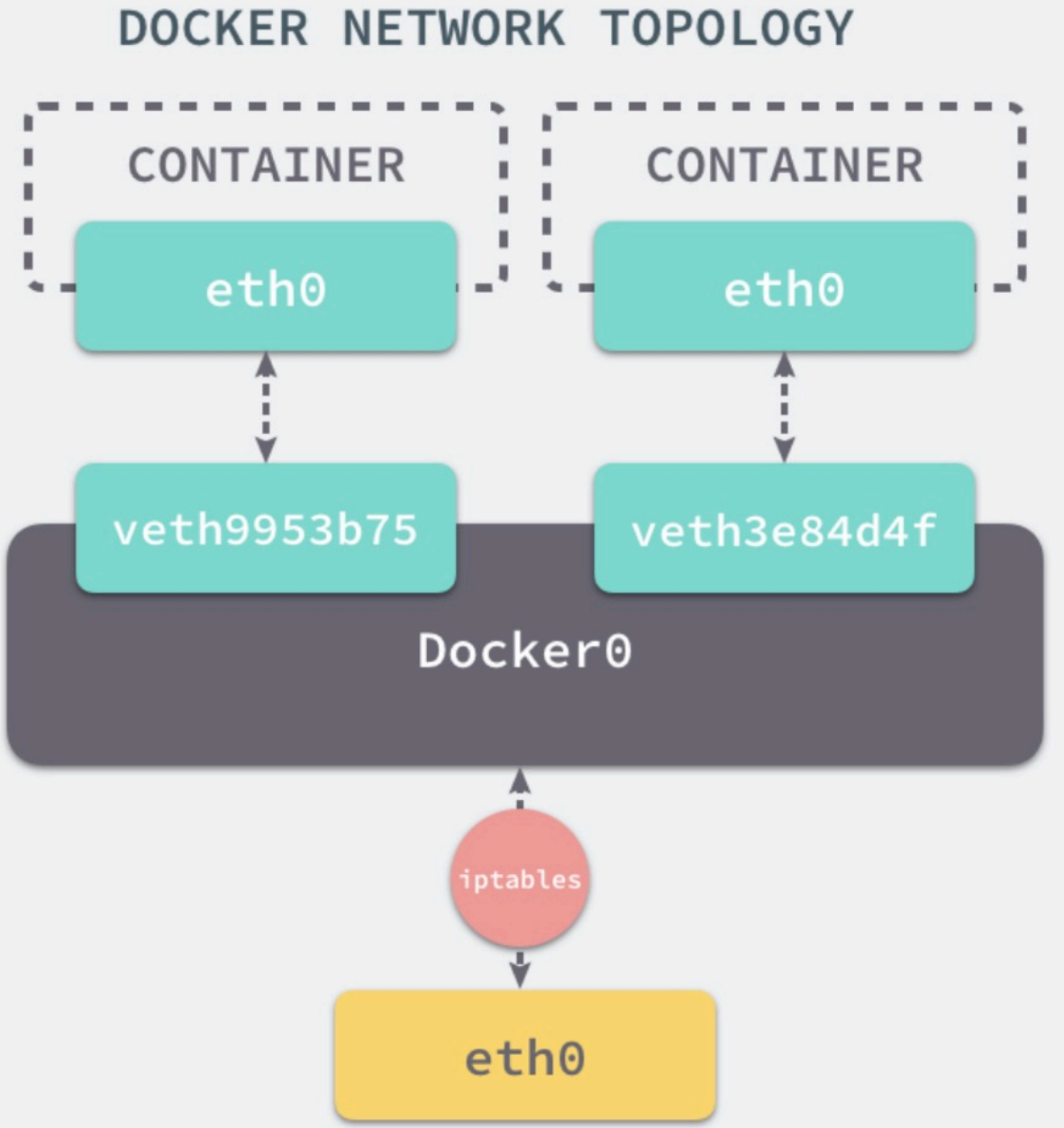
上述规则会将从*任意源发送到当前机器 6379 端口的 TCP 包*转发到 ***172.17.0.2:6379*** 所在的地址上。
telnet或ping 172.17.0.2都是通的：

```
$ telnet 172.17.0.2 6379
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
```

Docker网络模式 - Bridge 模式

Bridge模式示意图





Docker网络模式 - Bridge 模式

启动Docker容器前后， 查看iptables规则的变化

请参考这里

对比启动Nginx容器之后， 当前 iptables的NAT 配置在 DOCKER 链中出现了一条新规则：

```
16a17
> MASQUERADE  tcp  --  172.17.0.2       172.17.0.2       tcp dpt:6379
22a24
> DNAT       tcp  --  anywhere         anywhere         tcp dpt:6379 to:172.17.0.2:6379
```

上述规则会将从*任意源发送到当前机器 6379 端口的 TCP 包*转发到 ***172.17.0.2:6379*** 所在的地址上。
telnet或ping 172.17.0.2都是通的：

```
$ telnet 172.17.0.2 6379
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
```

Docker网络模式 - Bridge 模式

当使用 redis-cli 在宿主机器的命令行中访问 127.0.0.1:6379 的地址时，*经过 iptables 的 NAT PREROUTING* 将 ip 地址定向到了 172.0.0.2，重定向过的数据包就可以*通过 iptables 中的 FILTER* 配置，最终*在 NAT POSTROUTING* 阶段将 ip 地址伪装成 127.0.0.1，到这里虽然从外面看起来我们请求的是 127.0.0.1:6379，但是实际上请求的已经是 Docker 容器暴露出的端口了。

再次查看网卡信息，有个docker0网卡，以及一个veth虚拟网口:

```
$ brctl show
bridge name bridge id       STP enabled interfaces
br-291136fc6664     8000.02424b136faa   no
br-a32bb3ce11bf     8000.0242e0752eba   no
docker0     8000.02420b2e6681   no      vethe72c0bc
```

Docker网络模式 - Bridge 模式

容器与宿主机的通信

在桥接模式下，Docker Daemon ***将 veth0 附加到 docker0 网桥***上，保证宿主机的报文有能力发往 veth0。再将 ***veth1 添加到 Docker 容器所属的网络命名空间***，保证宿主机的网络报文若发往 veth0 可以立即被 veth1 收到。

Docker网络模式 - Bridge 模式

容器与外界通信

在桥接模式下，Docker Daemon *将 veth0 附加到 docker0 网桥*上，保证宿主机的报文有能力发往 veth0。再将 ***veth1 添加到 Docker 容器所属的网络命名空间***，保证宿主机的网络报文若发往 veth0 可以立即被 veth1 收到。

Docker网络模式 - Bridge 模式

容器与外界通信

默认情况下，容器可以访问到外部网络，但是外部网络无法访问到容器。容器要想访问外部网络，需要*本地系统的转发支持*。在Linux 系统中，检查转发是否打开。

```
$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1
```

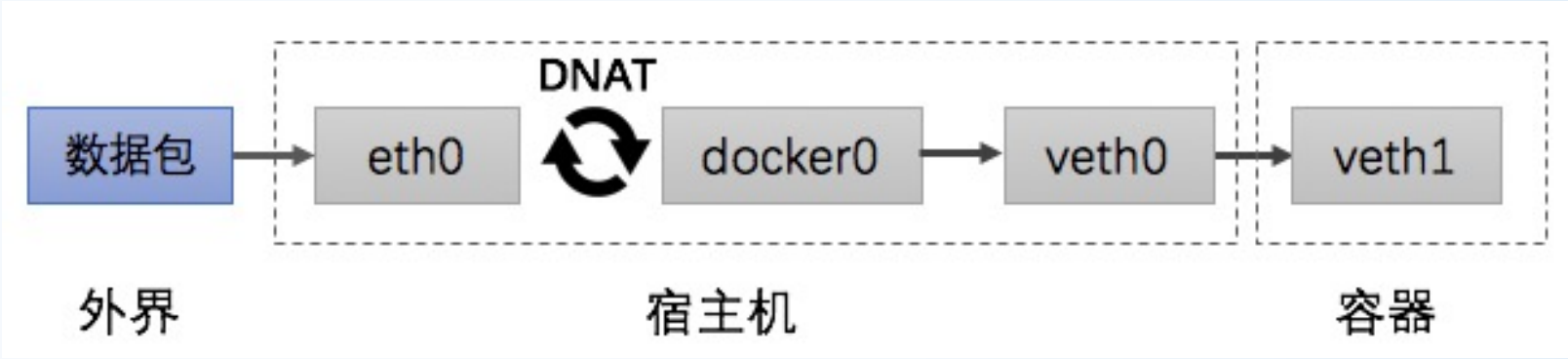
如果为 0，说明没有开启转发，则需要手动打开。

```
$ sysctl -w net.ipv4.ip_forward=1
```

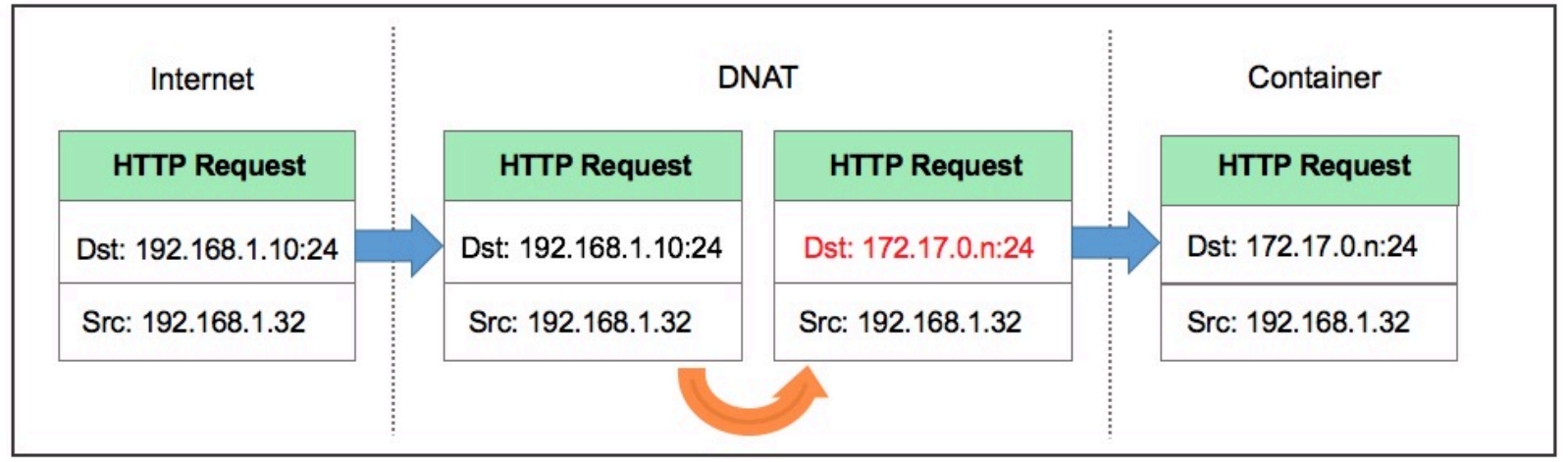
Docker网络模式 - Bridge 模式

容器与外界通信：由外到内

- 容器如果需要联网，则需要采用 NAT 方式。准确的说，是 NATP (网络地址端口转换) 方式。NATP 包含两种转换方式：SNAT 和 DNAT 。



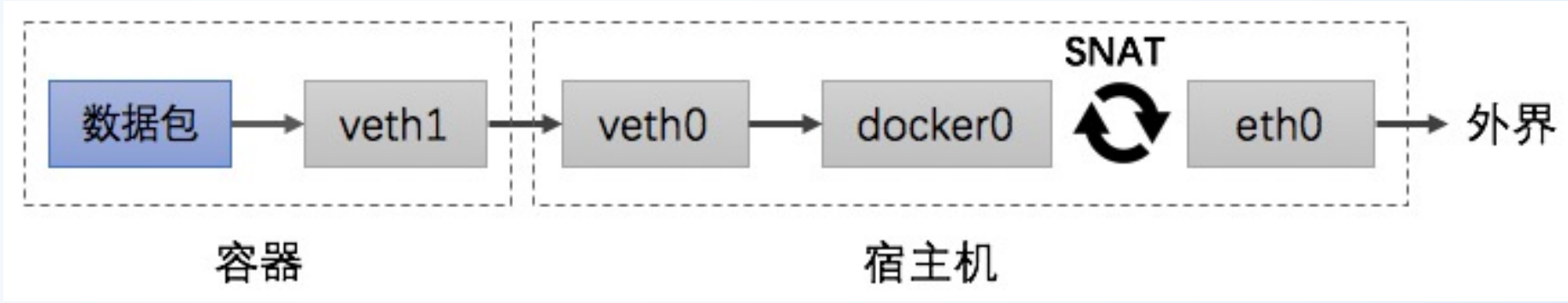
- 目的 NAT (Destination NAT，DNAT): 修改数据包的目的地址，当宿主机以外的世界需要访问容器时，数据包的流向如右图：

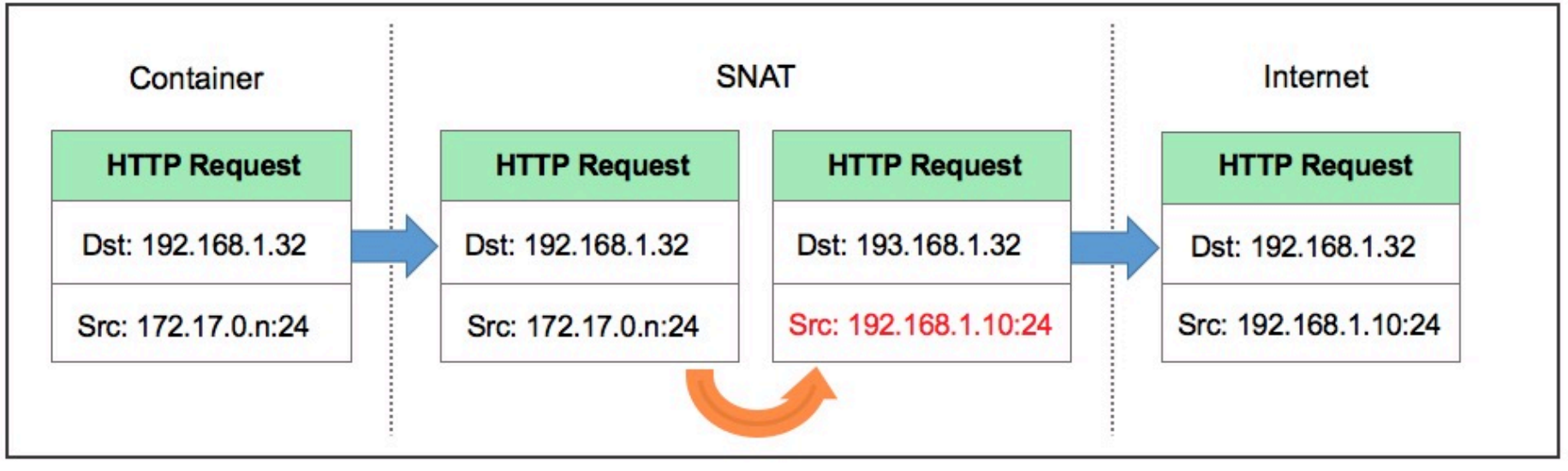


Docker网络模式 - Bridge 模式

容器与外界通信：由内到外

源 NAT (Source NAT, SNAT): 修改数据包的源地址。
当容器需要访问宿主机以外的世界时，数据包的流向如右图：

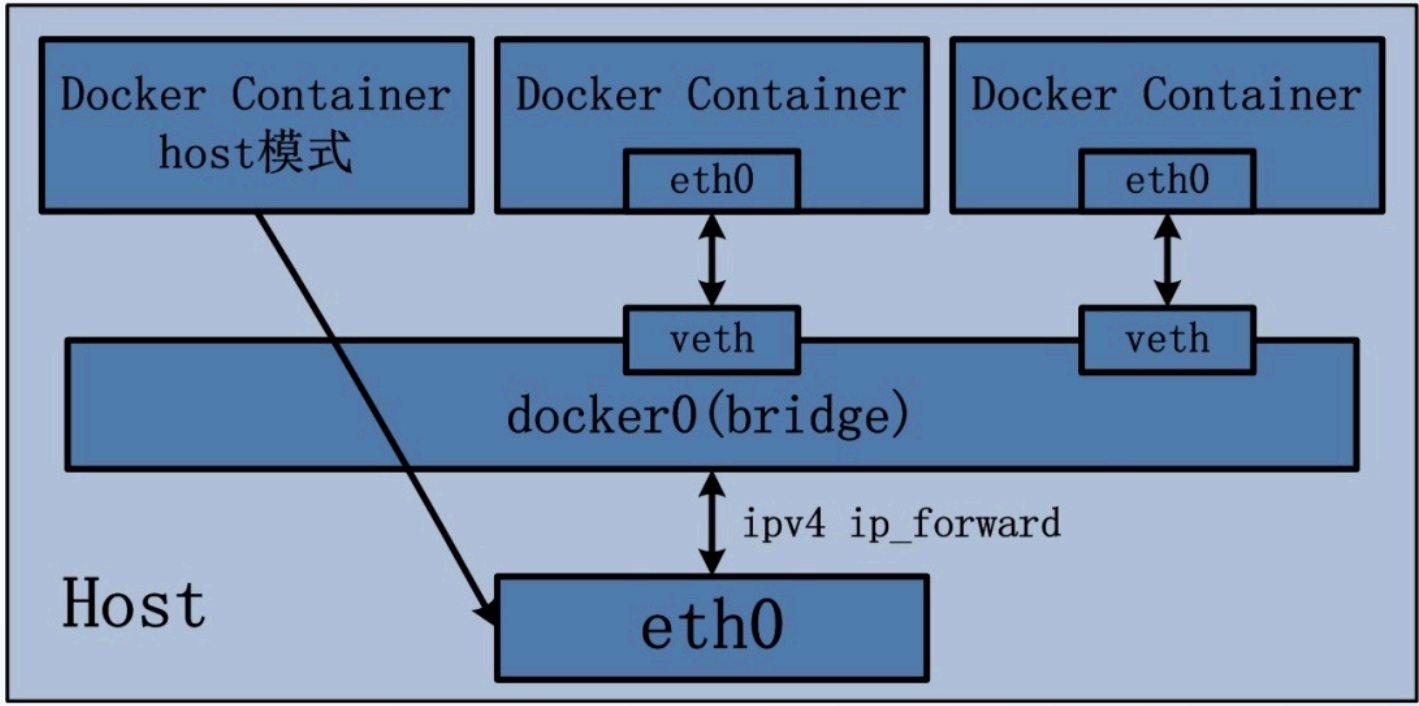




Docker网络模式 - Host网络模式

Host网络模式

Host模式并没有为容器创建一个隔离的网络环境。而之所以称之为host模式，是因为该模式下的Docker Container会和host宿主机*共享同一个网络namespace，拥有与主机相同的网络设备*，故Docker Container可以和宿主机一样，*使用宿主机的eth0，实现和外界的通信*。换言之，Docker Container的IP地址即为*宿主机eth0的IP地址*。



Docker网络模式 - Host网络模式

Host网络模式

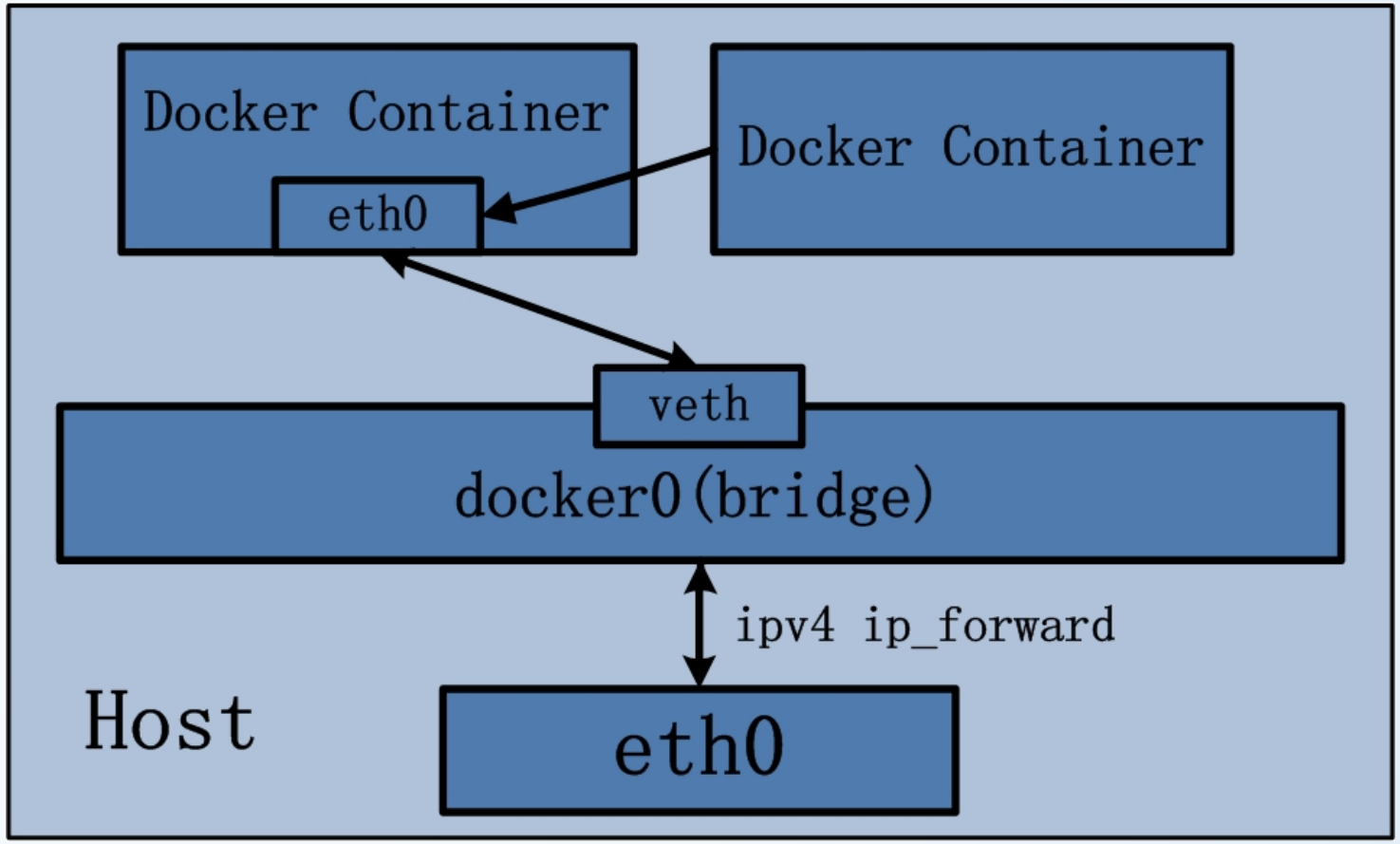
- 创建host网络模式的Container非常简单，只需用*--net=host* 即可。
- 通过*ip addr*查看，与宿主机的信息完全系统，因为直接利用宿主机的网络栈，所以发现并没有veth信息。

```
$ docker run --net=host -it busybox  ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    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 mq qlen 1000
    link/ether 94:57:a5:55:be:24 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.127/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000
    link/ether 94:57:a5:55:be:25 brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000
    link/ether 94:57:a5:55:be:26 brd ff:ff:ff:ff:ff:ff
5: eth3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000
    link/ether 94:57:a5:55:be:27 brd ff:ff:ff:ff:ff:ff
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
    link/ether 02:42:0b:2e:66:81 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
7: br-a32bb3ce11bf: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
    link/ether 02:42:e0:75:2e:ba brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.1/16 scope global br-a32bb3ce11bf
       valid_lft forever preferred_lft forever
```

Docker网络模式 - Container模式网络

Container模式网络

该模式会*重用另一个容器的网络命名空间*。通常来说，当你想要自定义网络栈时，该模式是很有用的。实际上，该模式也是***Kubernetes使用网络***模式。



Docker网络模式 - Container模式网络

Container模式网络

启动一个bridge模式的容器：

$ docker run -itd --rm --name c1 -d -P --net=bridge busybox:latest sleep 120

启动busybox container，指定 * –net=container:nginx * 指定只用 nginx容器网络

$ docker run -itd --rm --name c2 --net=container:nginx busybox:latest sleep 120

查看IP地址：

$ docker exec -it c1 ip addr

$ docker exec -it c2 ip addr

采用container模式的2个容器共享相同的1个network namespace

Docker网络模式 - none模式

顾名思义，网络环境为none，即不为Docker Container任何的网络环境。一旦Docker Container采用了none网络模式，那么容器内部就*只能使用loopback网络设备*，不会再有其他的网络资源。

可以说none模式为Docker Container做了极少的网络设定，但是俗话说得好“少即是多”，在没有网络配置的情况下，作为Docker开发者，才能在这基础做其他无限多可能的*网络定制开发*。

这也恰巧体现了Docker设计理念的开放。实际上，该模式关闭了容器的网络功能。在以下两种情况下是有用的：

- 容器并*不需要网络*（例如只需要写磁盘卷的批处理任务）
- 希望*自定义网络*。

Docker网络模式 - none模式

通过* --net=none*启动一个container

$ docker run --name=c1 --net=none -it --rm busybox ip addr

可看到仅有一个lo换回地址：

```
$ docker exec -it 76 ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    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
```

Docker网络模式 - overlay网络模型

overlay网络模型

当前Docker跨主机网络方案很多，比如：

- docker 原生的 overlay
- flannel、weave 和 calico

而Docker ***通过 libnetwork 以及 CNM*** 将上述各种方案与docker集成在一起。
libnetwork 是 docker 容器网络库，最核心的内容是其定义的 Container Network Model (CNM)，这个模型对容器网络进行抽象，由以下三类组件组成：

- *Endpoint*
- *Network*
- *Sandbox*

Docker网络模式 - overlay网络模型

- **Sandbox：**

Sandbox 是*容器的网络栈*，包含容器的 interface、路由表和 DNS 设置。 Linux Network Namespace 是 Sandbox 的标准实现。Sandbox 可以包含来自不同 Network 的 Endpoint。也就是说***Sandbox将一个容器与另一个容器通过Namespace进行隔离***，一个容器包含一个sandbox，每一个sandbox可以有多个Endpoint隶属于不同的网络。

- **Endpoint：**

Endpoint 的作用是将 Sandbox 接入 Network。 Endpoint ***的典型实现是 veth pair***。一个 Endpoint 只能属于*一个网络*，也只能属于***一个 Sandbox***。

- **Network**

Network 包含一组 Endpoint，同一 Network 的 ***Endpoint 可以直接通信***。 Network 的实现可以是 Linux Bridge、VLAN 等。

Docker网络模式 - overlay网络模型

配置Docker overlay 网络

请参见：配置Docker overlay 网络例子

overlay网络原理

overlay 网络数据还是从 ***bridge 网络docker_gwbridge***出去的，但是由于Consul的作用（记录了overlay网络的endpoint、sandbox、network等信息），使得Docker知道了此网络是overlay 类型的，这样此overlay网络下的不同主机之间就能够相互访问，但其实出口还是在docker_gwbridge网桥。

课程回顾

已学知识要点

- *Docker网络模型*
- *容器与外界通信*
- *跨宿主机网络*