4、Docker容器相关命令

- 容器核心概念

> docker镜像（类似我们的安装包）->通过docker的命令运行镜像（类似安装好的软件进行启动）->产生一个容器（正在运行的独立程序，可停止，可再次启动，不需要时可删除掉容器，每次运行镜像都会产生一个独立的容器，可自定义容器名）

操作	命令	说明
运行	docker run --name container-name -d image-name eg:docker run --name myredis -d redis	--name：自定义容器名 -d：后台运行 image-name:指定镜像模板
列表	docker ps（查看运行中的容器）；	加上-a；可以查看所有容器
停止	docker stop container-name/container-id	停止当前你运行的容器
启动	docker start container-name/container-id	启动容器
删除	docker rm container-id	删除指定容器
端口映射	-p 6379:6379 eg:docker run -d -p 6379:6379 --name myredis docker.io/redis	-p: 主机端口(映射到)容器内部的端口
容器日志	docker logs container-name/container-id	
更多命令	https://docs.docker.com/engine/reference/commandline/docker/	

- 查看容器

```
docker ps //查看运行中的容器
docker ps -a//查看所有的容器（包括所有不在运行的容器）
```

- 重启容器

```
docker restart container-id
```

- 删除容器

```
docker rm container-id或容器名字
```

- 使用一个镜像可以运行出多个独立的容器
  - --name：后面指定容器的名字，不可重复，（不指定的话会生成一个随机的名字）
  - -d：后台运行
  - tomcat:8-jre8：镜像名
  - -p：端口映射，8887对外暴露的端口，8080是映射到docker容器的提供服务的端口；
  - -P：不指定端口会给定随机端口
  - -e：指定运行时的一些配置

```
docker run --name mytomcat1 -d -p 8888:8080 tomcat:8-jre8
docker run --name mytomcat2 -d -p 8887:8080 tomcat:8-jre8
docker run -p 3306:3306 --name mysql1 -e MYSQL_ROOT_PASSWORD=123456 -d mysql
```

- 查看容器中进程信息

```
docker top 容器名或容器ID
```

- 挂载目录
  - 将容器中的文件挂载到宿主机上，通过echo的方式修改宿主机的文件(vim方式是不生效的)，达到自动修改容器中文件的目的。具体参见：笔记6-1、6-2。

- 查看docker容器的启动日志

```
docker logs 参数 容器ID
    参数：
    -f follow 表示实时显示日志
    -t timestamp 表示显示时间戳
    --tail=n 或 --tail n ，n表示显示末尾n行

docker logs -f --tail=200 容器ID，表示实时加载日志信息，并且仅显示最后200行
```

- 查找容器的启动日志文件中含有特定字符串的行,并且可以输出到指定文件

```
docker logs 参数 容器id | grep str
docker logs 参数 容器id | grep str >> out.txt
```

- 查找某个时间范围的容器启动日志

```
--since 从指定时间点到最新的日志
--until 指定结束时间点
docker logs --since 2020-04-10T19:50:00 container
docker logs --since 2020-04-10T19:30:00 --until 2020-04-10T20:05:00 contailer
```

- 启动报错：max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144

```
#切换到root用户
    su root
#编辑 /etc/sysctl.conf
    vi  /etc/sysctl.conf
 #添加如下参数
    vm.max_map_count=2621441
 #执行如下命令，设置永久改变
    sudo sysctl -p /etc/sysctl.conf
```

- 启动报错：max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536]

```
切换到root用户 然后强制修改/etc/security/limits.conf文件，添加或修改如下行：
*       hard    nofile          65536
*       soft    nofile          65536
```

- 进入容器内部的命令行
  - -it就等于 -i和-t，这两个参数的作用：i:打开容器的标准输入，t:告诉docker为容器建立一个命令行终端，所以就是为该docker创建一个伪终端，这样就可以进入到容器的交互模式（也就是直接进入到容器里面）
  - /bin/bash的作用：告诉docker要在容器里面执行此命令，表示载入容器后运行bash；因为docker中必须要保持一个进程的运行，要不然整个容器启动后就会马上kill itself，这个/bin/bash就表示启动容器后启动bash
  - 而通过exit或Ctrl+P+Q可以退出当前容器，到linux系统

```
//开始运行的同时进入容器内部
docker run -it --name 容器名  -d -p 外部端口:内部端口 镜像名  /bin/bash

//进入运行中的容器的内部（进入容器后，开启一个新的终端，可以在里面操作）
 docker exec -it 容器名或容器ID /bin/bash

////进入运行中的容器的内部（进入容器正在执行的终端，不会开启一个新的终端）
docker attach 容器名或容器ID
```

- 将容器内部的文件拷贝到宿主机上

```
docker cp 容器id:容器内路径   宿主机的目的的路径
```

```
# 进入docker容器内部
[root@kuangshen home]# docker attach b78453025116
[root@b78453025116 /]# cd /home
[root@b78453025116 home]# ls
# 在容器内新建一个文件
[root@b78453025116 home]# touch test.java
[root@b78453025116 home]# exit
exit
[root@kuangshen home]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
[root@kuangshen home]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS
    PORTS               NAMES
b78453025116        centos              "/bin/bash"         About a minute ago   Exited (0) 7 seconds
ago                       frosty_hertz

# 将这文件拷贝出来到主机上
[root@kuangshen home]# docker cp b78453025116:/home/test.java /home
[root@kuangshen home]# ls
!   idea  kuangshen  kuangshen.java  mysql  test.java
[root@kuangshen home]#

# 拷贝是一个手动过程，未来我们使用 -v 卷的技术，可以实现，自动同步
```

拷贝操作是在宿主机上操作的

- 解决外部访问不到docker容器的问题
  - docker容器启动时，会自动分配一个docker0的虚拟网卡(默认是桥接模式bridge)，所以在运行出容器后，通过外部访问容器时，需要打开ip_forward转发规则来指定宿主机需要绑定的虚拟网卡docker0，外部才能通过访问宿主机ip转发到docker0虚拟网卡的容器（是docker0虚拟网卡的子ip），如果不打开ip_forward转发规则，只能在运行出容器时，指定参数使用host主机网络模式，因为使用（--net=host，主机模式）主机模式可以共享使用宿主机的网卡eth0的ip，外部可以直接通过宿主机ip访问。
  - 原理讲解
    - [(3条消息) docker 启动时指定需要绑定的网卡_Docker容器网络-基础篇_蒋咔咔的博客-CSDN博客]
    - 见下方网络模式详细解释
  - 原因
    - 外部（使用宿主机ip+映射端口）访问时，需要先关闭防火墙(切换到bin目录下)，这里仅使用前两个命令

        - service firewalld status或firewall-cmd --state
        - systemctl stop firewalld.service或service firewalld stop
        - systemctl start firewalld.service
    - 因为docker容器启动自动分配一个虚拟网卡所在网段的非占用的新ip，所以需要配置ip转发，外部ip才能找到此容器的ip
- 临时方案：这种方式无需重启docker容器，不过也只能算是临时生效，它的效果会随着计算机的重启而失效。

```
//查看ip转发是否打开（0未打开  1打开）
      sysctl net.ipv4.ip_forward  或  cat /proc/sys/net/ipv4/ip_forward
//修改 net.ipv4.ip_forward 的值为1
      sysctl -w net.ipv4.ip_forward=1 或  echo 1 > /proc/sys/net/ipv4/ip_forward
```

- 临时方案

```
在运行容器时，指定  --ip-forward=true 也可以临时开启ip包转发功能
```

- 永久方案（需重启docker容器）

```
#修改配置文件
vim /etc/sysctl.conf文件，新增 "net.ipv4.ip_forward = 1"，保存退出
# 立即生效
sysctl -p /etc/sysctl.conf
# 并且重启网卡
redhat系列：service network restart 或 systemctl restart network
debian/ubuntu系列：/etc/init.d/procps.sh restart
#重启容器
systemctl restart 容器ID
```

- 查看当前容器的端口

```
docker port 容器id
```

- 自动进行空间清理

```
docker system prune
```

- 查看容器的内存占用

```
docker stats
```

- 安装docker服务的时候，会自动安装三种基本网络（bridge, host, none）和自定义网络，自定义可选的模式有：bridge、overlay、macvlan。
- 查看docker服务中有哪些网络模式(默认是bridge网络模式)
  - 参考：https://blog.csdn.net/gezhonglei2007/article/details/51627821

```
docker network ls //查看所有网络
docker network create //创建一个网络
docker network connect//连接一个网络
docker network rm //移除一个网络
docker network disconnect //断开一个网络
docker network inspect //查看一个网络
```

- 查看bridge网络模式（包括网络的信息及此网络下所有正在运行的容器信息）

```
docker network inspect bridge
```

- 网络模式详细解释
  - 参考：
  - https://blog.csdn.net/suchahaerkang/article/details/84570488
  - https://www.cnblogs.com/goloving/p/15133673.html

（1）四种网络模式

Bridge：此模式会为新创建的每一个容器分配独立的IP，并将所有容器连接到一个docker0虚拟网桥（同一个网段下），通过docker0虚拟网桥以及Iptables nat表(ip地址转换)来配置与宿主机通信，则通过宿主机ip能找到每个容器。

host：新创建的容器将不会拥有一个独立的网卡和IP，而是使用宿主机的网卡和IP、端口。但是，容器的其他方面，如文件系统、进程列表等还是和宿主机隔离的。

Container：新创建的容器不会创建自己的网卡和IP，而是和一个已经存在的容器共享网卡和IP、端口范围，而不是和宿主机共享。两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。两个容器的进程可以通过 lo 网卡设备通信。

None：该模式关闭了容器的网络功能，Docker容器拥有自己的Network Namespace，但是并不为Docker容器进行任何网络配置，也就是说，这个Docker容器没有网卡、IP、路由等信息。需要我们自己为Docker容器添加

网卡、配置IP等。

（2）Bridge网络模式

基本流程

（1）当Docker server启动时，会在主机上创建一个名为docker0的虚拟网桥，并且会从RFC1918所定义的私有IP网段中随机选择一个和宿主机不同的IP地址和子网分配给docker0虚拟网桥，虚拟网桥docker0在内核层连通了其他的物理或虚拟网卡，而启动的Docker容器都会连接到这个docker0虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，主机上的所有容器就通过docker0虚拟网桥连在了一个二层虚拟网络中。接下来就要为容器分配IP了，连接到docker0虚拟网桥的容器就从这个子网中选择一个未占用的IP使用。

（2）比如：一般会使用172.17.0.0/16这个子网（网段），并将172.17.0.1/16分配给docker0虚拟网桥（在主机上使用ifconfig或ip addr命令是可以看到docker0虚拟网桥，可以认为它是虚拟网桥的管理接口，在宿主机上作为一块虚拟网卡使用），而创建一个容器时，都使用此网段未占用的IP地址：172.17.0.x，并使用docker0的IP地址172.17.0.1作为容器的默认网关，因为在同一宿主机内的容器都接入同一个网桥docker0，这样容器之间就能够通过容器的Container-IP直接通信。docker0网桥是宿主机虚拟出来的，即docker0虚拟网桥和宿主机的 eth0 网卡不处于同一个网段，docker0网桥并不是真实存在的网络设备，外部网络是无法寻址到的，这也意味着外部网络无法通过直接Container-IP访问到容器。如果容器希望外部访问能够访问到，需要配置docker0虚拟网桥和宿主机能通信即可，将iptable 转发打开，并且通过映射容器端口到宿主主机（端口映射），即docker run创建容器时候通过 -p 或 -P 参数来启用，访问容器的时候就通过[宿主机IP]:[映射端口]访问容器。

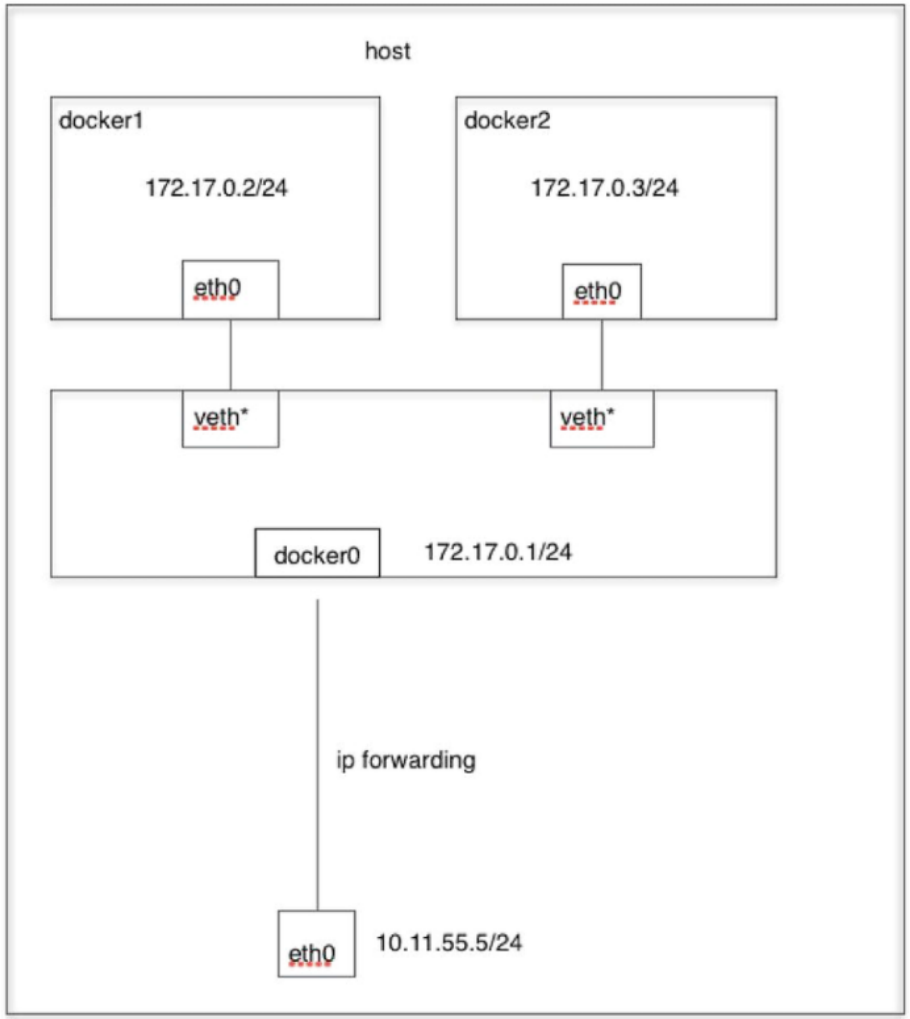
内部通信原理

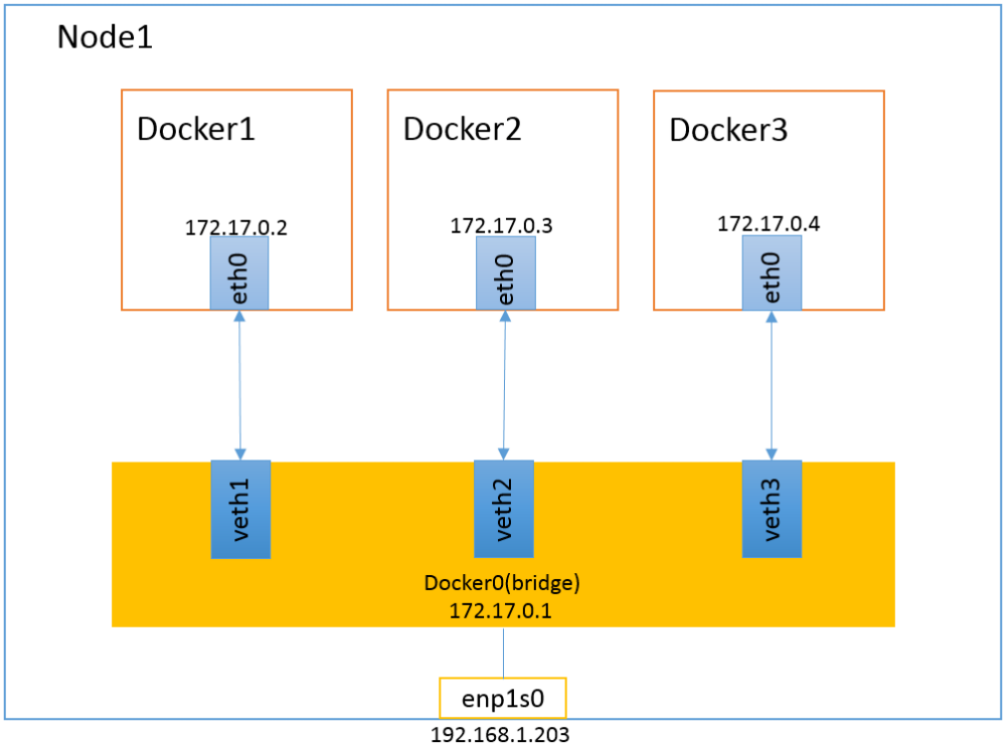
（1）Docker Daemon 利用 veth pair 虚拟设备技术（下图中的连接线两端），它们组成了一个数据的通道，数据从一个设备进入，就会从另一个设备出来。因此，veth设备常用来连接两个网络设备。

（2）Docker Daemon 将 veth pair虚拟设备的一端放在宿主机中，以 vethxx 这样类似的名字命名，并将这个网络设备加入到docker0网桥中，可以通过brctl show命令查看。

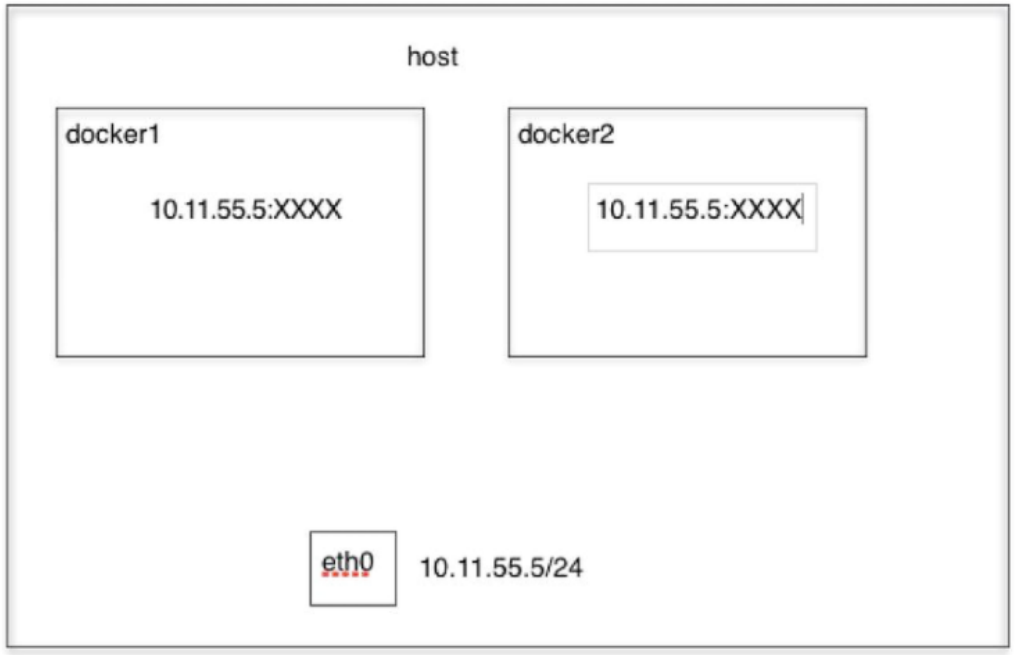
（3）Docker Daemon veth pair将虚拟设备的另一端添加到 Docker Container 所属的 namespace 下,并被改名为 eth0。这样就实现宿主机到Docker Container 网络的联通性;同时,也保证 Docker Container 网络环境的隔离性。

- Bridge

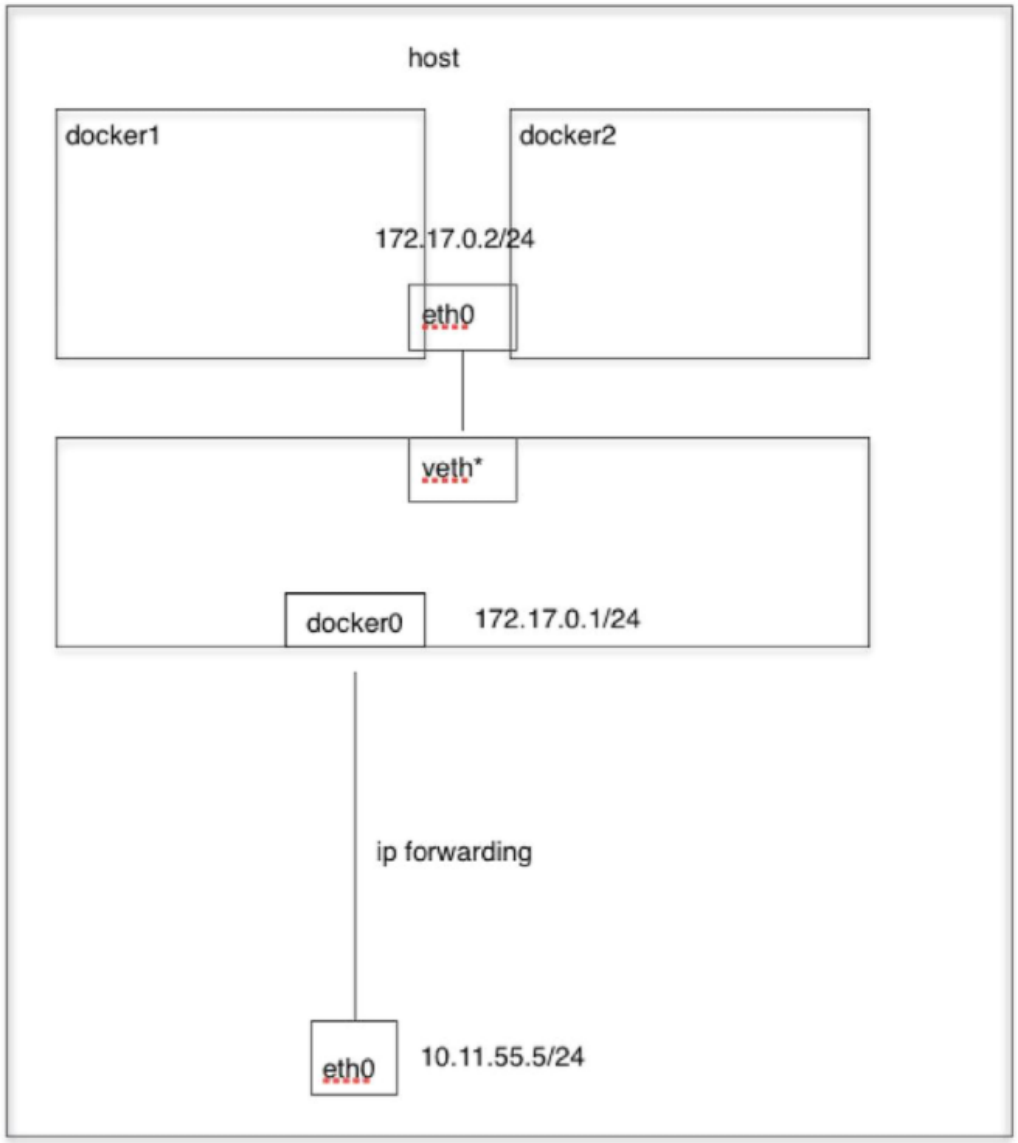

host
docker1
172.17.0.2/24
eth0
docker2
172.17.0.3/24
eth0
veth*
veth*
docker0
172.17.0.1/24
ip forwarding
eth0
10.11.55.5/24


Node1
Docker1
172.17.0.2
eth0
Docker2
172.17.0.3
eth0
Docker3
172.17.0.4
eth0
veth1
veth2
veth3
Docker0(bridge)
172.17.0.1
enp1s0
192.168.1.203

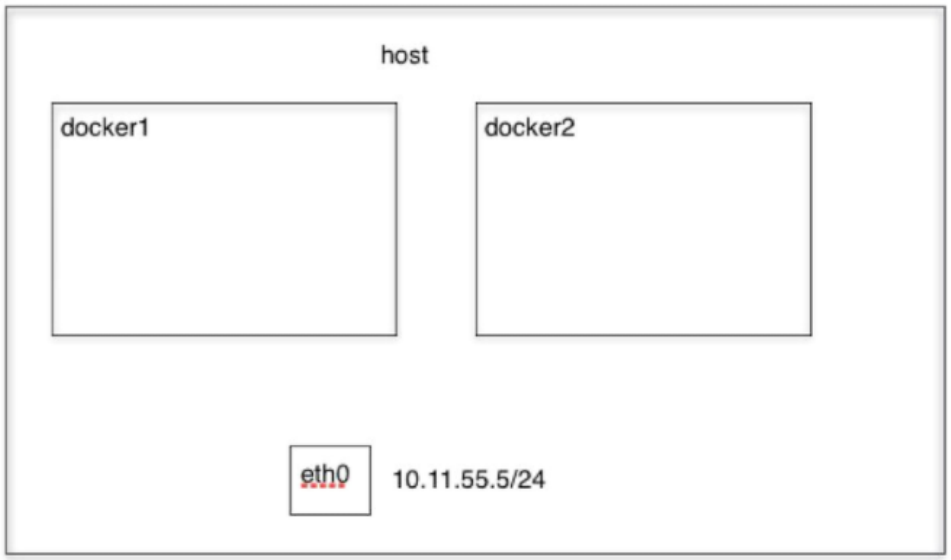
- host


host
docker1
10.11.55.5:XXXX
docker2
10.11.55.5:XXXX
eth0
10.11.55.5/24

- Container


host
docker1
docker2
172.17.0.2/24
eth0
veth*
docker0
172.17.0.1/24
ip forwarding
eth0
10.11.55.5/24

- none


host
docker1
docker2
eth0
10.11.55.5/24