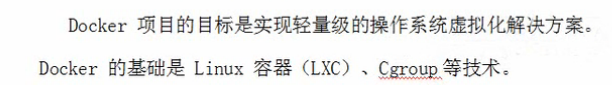
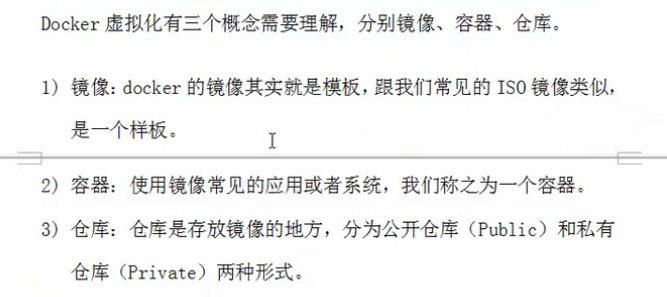
虚拟化

Kvm

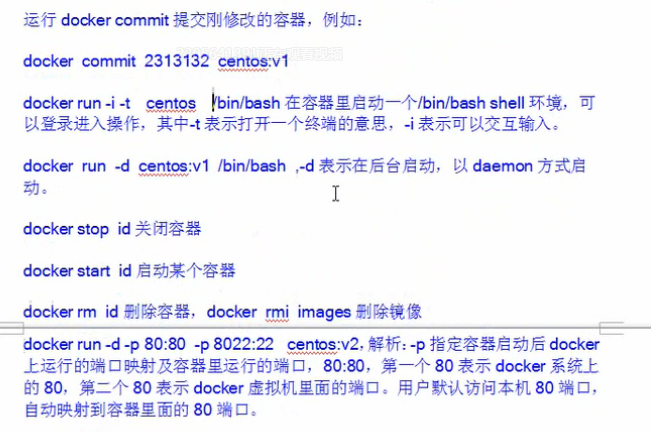
Xen





常用命令





停用全部运行中的容器:

# docker stop $(docker ps -q)

删除全部容器：

# docker rm $(docker ps -aq)

要区分容器的ID和镜像的ID

进入docker容器后，如果输入了exit退出容器，我们可以在使用docker start id来启动容器，在使用docker attach连接进去。

ctrl-p 再按 ctrl-q ，这样就可以退出到宿主机

# docker ps --help

可以查看具体的命令

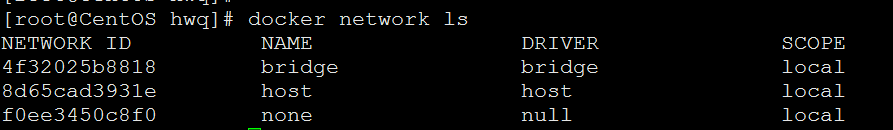
Docker 网络配置

<http://www.cnblogs.com/gispathfinder/p/5871043.html>

<https://docs.docker.com/engine/userguide/networking/>

官方文档

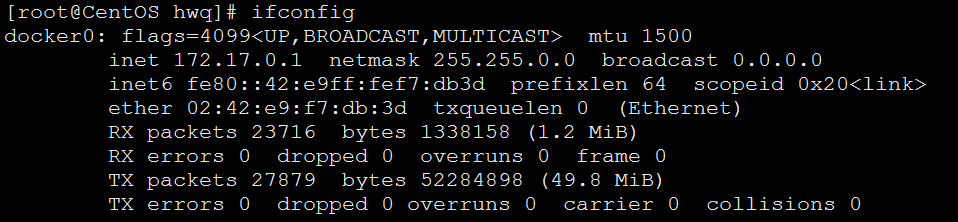
Docker安装后会自动创建3个网络，可以使用 docker network ls 查看一下



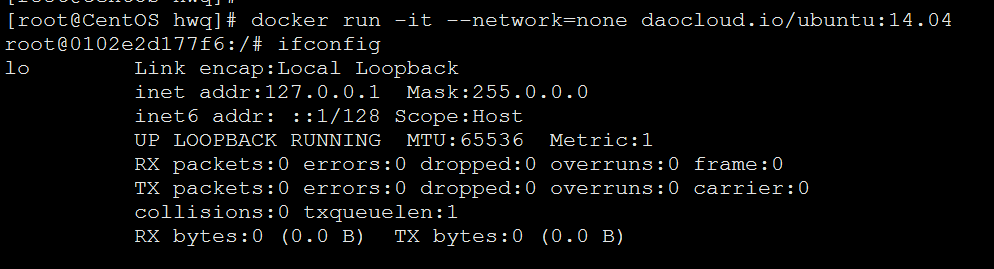
当你启动容器的时候，你可以使用 --networek 参数来指定你需要的网络类型来启动。

Docker默认使用 bridge 网络来进行连接 （docker0） ，你可以使用 docker run --network=<NETWORK> 来指定其它网络。

* Bridge网络模式



* None网络模式，容器有容器指定的网络协议栈，但是并没有进行网络的额皮质。



* Host网络模式：与宿主机的网络配置相同。

Docker默认使用bridge网络模式，我们可以使用 docker network inspect来查看



一、Docker提供了四种网络模式：

- host模式，使用–net=host指定。

- container模式，使用–net=container:NAME\_or\_ID指定。

- none模式，使用–net=none指定。

- bridge模式，使用–net=bridge指定，默认设置。

下面分别简单介绍一下各种网络模式。

* host模式

如果启动容器的时候使用host模式，那么这个容器将不会获得一个独立的Network Namespace，而是和宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡，配置自己的IP等，而是使用宿主机的IP和端口。

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模式

Container模式指定新创建的容器和已经存在的一个容器共享一个Network Namespace，而不是和宿主机共享。新创建的容器不会创建自己的网卡，配置自己的IP，而是和一个指定的容器共享IP、端口范围等。同样，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模式

使用none模式，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。

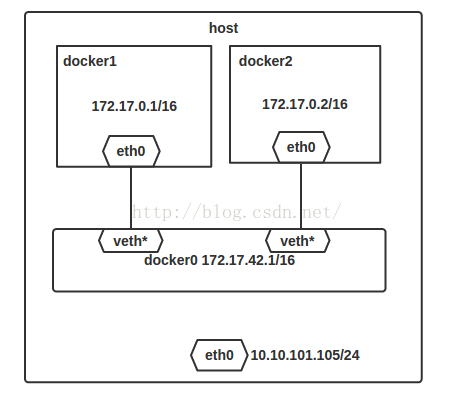
* bridge模式

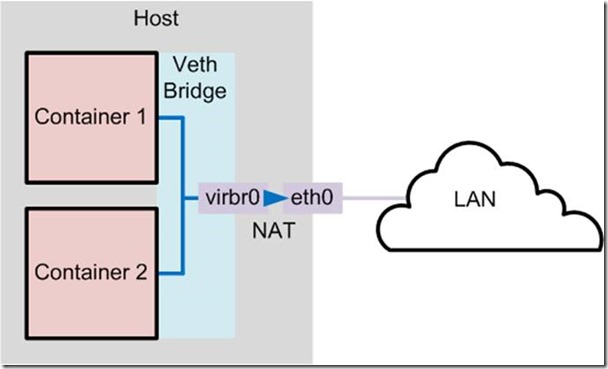
bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将一个主机上的Docker容器连接到一个虚拟网桥上。

**桥接网络详解：**

当Docker server启动时，会在主机上创建一个名为docker0的虚拟网桥，此主机上启动的Docker容器会连接到这个虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，这样主机上的所有容器就通过交换机连在了一个二层网络中。

桥接网络拓扑结构如下



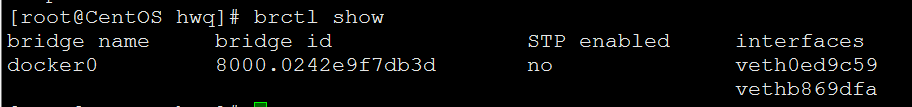


Docker完成以上网络配置的过程大致是这样的：

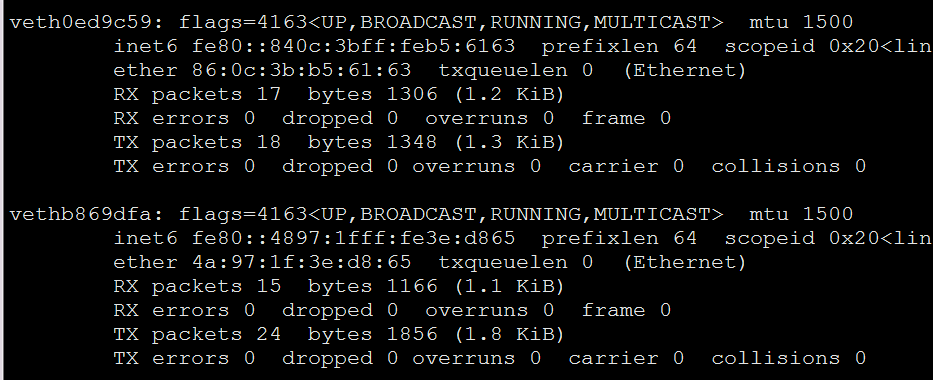
1. 在主机上创建一对虚拟网卡veth pair设备。veth设备总是成对出现的，它们组成了一个数据的通道，数据从一个设备进入，就会从另一个设备出来。因此，veth设备常用来连接两个网络设备。

2. Docker将veth pair设备的一端放在新创建的容器中，并命名为eth0。另一端放在主机中，以veth65f9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。

（yum install bridge-utils）



网络ifconfig



3. 从docker0子网中分配一个IP给容器使用，并设置docker0的IP地址为容器的默认网关。

**bridge模式下容器的通信**

在bridge模式下，连在同一网桥上的容器可以相互通信（若出于安全考虑，也可以禁止它们之间通信，方法是在DOCKER\_OPTS变量中设置--icc=false，这样只有使用--link才能使两个容器通信）。

容器也可以与外部通信，我们看一下主机上的Iptable规则，可以看到这么一条

-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE

这条规则会将源地址为172.17.0.0/16的包（也就是从Docker容器产生的包），并且不是从docker0网卡发出的，进行源地址转换，转换成主机网卡的地址。这么说可能不太好理解，举一个例子说明一下。假设主机有一块网卡为eth0，IP地址为10.10.101.105/24，网关为10.10.101.254。从主机上一个IP为172.17.0.1/16的容器中ping百度（180.76.3.151）。IP包首先从容器发往自己的默认网关docker0，包到达docker0后，也就到达了主机上。然后会查询主机的路由表，发现包应该从主机的eth0发往主机的网关10.10.105.254/24。接着包会转发给eth0，并从eth0发出去（主机的ip\_forward转发应该已经打开）。这时候，上面的Iptable规则就会起作用，对包做SNAT转换，将源地址换为eth0的地址。这样，在外界看来，这个包就是从10.10.101.105上发出来的，Docker容器对外是不可见的。

那么，外面的机器是如何访问Docker容器的服务呢？我们首先用下面命令创建一个含有web应用的容器，将容器的80端口映射到主机的80端口。

docker run -d --name web -p 80:80 fmzhen/simpleweb

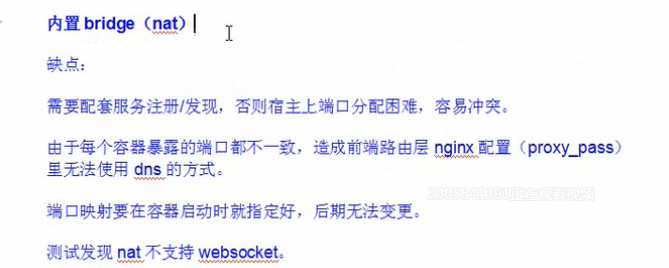
然后查看Iptable规则的变化，发现多了这样一条规则：

-A DOCKER ! -i docker0 -p tcp -m tcp --dport 80 -j DNAT --to-destination 172.17.0.5:80

此条规则就是对主机eth0收到的目的端口为80的tcp流量进行DNAT转换，将流量发往172.17.0.5:80，也就是我们上面创建的Docker容器。所以，外界只需访问10.10.101.105:80就可以访问到容器中得服务。

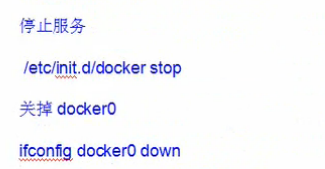
除此之外，我们还可以自定义Docker使用的IP地址、DNS等信息，甚至使用自己定义的网桥，但是其工作方式还是一样的。

Docker可以自定义自己的网桥

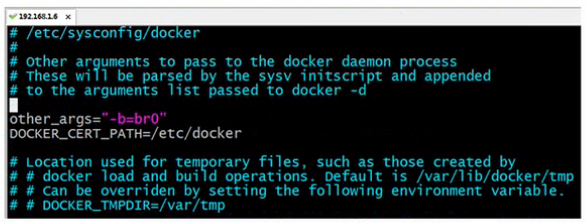


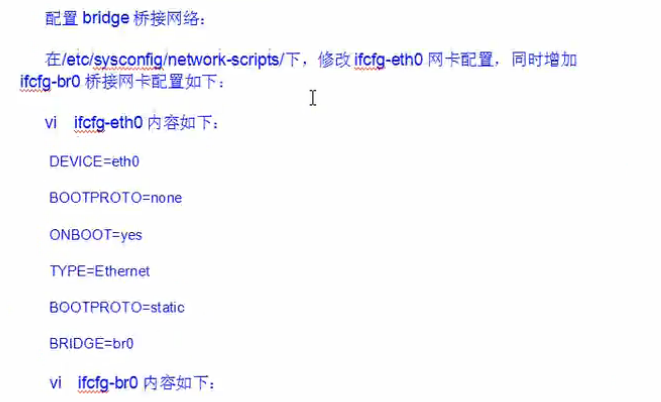


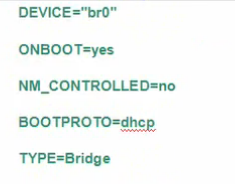
配置

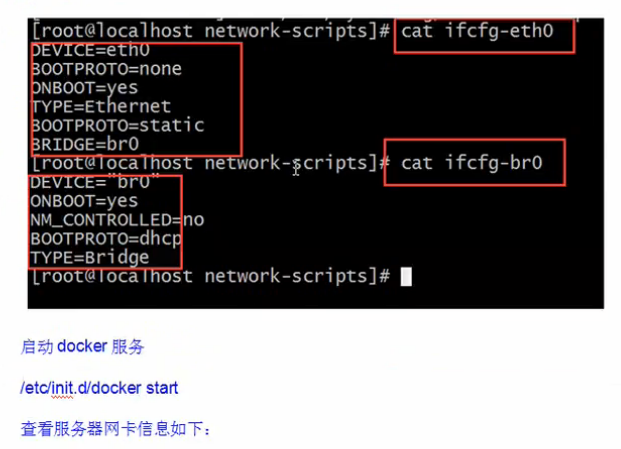














容器互联

<http://wiki.jikexueyuan.com/project/docker-technology-and-combat/linking.html>

网络映射

目录映射

Dockerfile

Docker权限

启动时设置参数 --privileged=true

使用该参数，Container内的root拥有真正的root权限。

否则，container内的root只是外部的一个普通用户权限。

Docker时间

# cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

<http://blog.csdn.net/u012373815/article/details/52782807>

<http://blog.csdn.net/wwq518/article/details/53318970>

<http://www.centoscn.com/CentOS/config/2015/0723/5901.html>