Docker网络设置

使用Docker run创建Docker容器是，可以用--net指定容器的网络模式，当前Docker有以下4中网络模式：

1）host模式，使用--net=host指定

2）container模式，使用--net=container.NAME\_OR\_ID指定

3）node模式，使用--net=none指定

4）bridge模式，使用--net=bridge指定，默认设置

# 1.host模式

Docker使用了Linux的Namespace技术来进行资源隔离，如PID Namespce隔离进程，Mount Namespace隔离文件系统，Network Namespace隔离网络等。一个Network Namespace提供一个独立的网络环境，包括网卡、路由、Iptable规则等都与其他Network Namespace隔离。一个Docker容器一般会分配一个独立的Network Namespace，如果启动时使用host模式，那么容器不会获得一个独立的Network Namespace，而是和宿主机公用一个Network Namespace，容器不会虚拟网卡、配置IP等，而是使用宿主的IP和端口，直接暴露在公共网络中。

指定方法：--net="host"

这样创建出来的容器，可以看到host上所有的网络设备，容器中，对这些设备（比如DUBS）有全部的访问权限，这种方式是不安全的，如果在隔离良好的环境中（比如租户的虚拟机中）使用这种方式，问题不大。

*[root@fys2 ~]# docker run -d -P tomcat*

*097255714cd5dae5b1af5d17b7967402fbabe797eb9d9404dfdc6b6158411172*

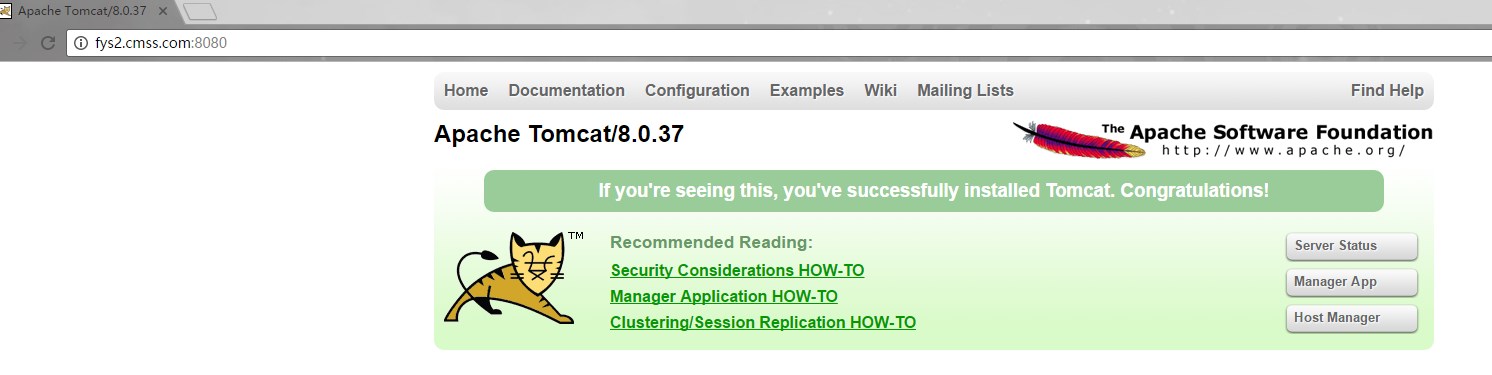
*[root@fys2 ~]# docker ps*

*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

*a20d251cde70 tomcat "catalina.sh run" 6 minutes ago Up About a minute hungry\_carson*

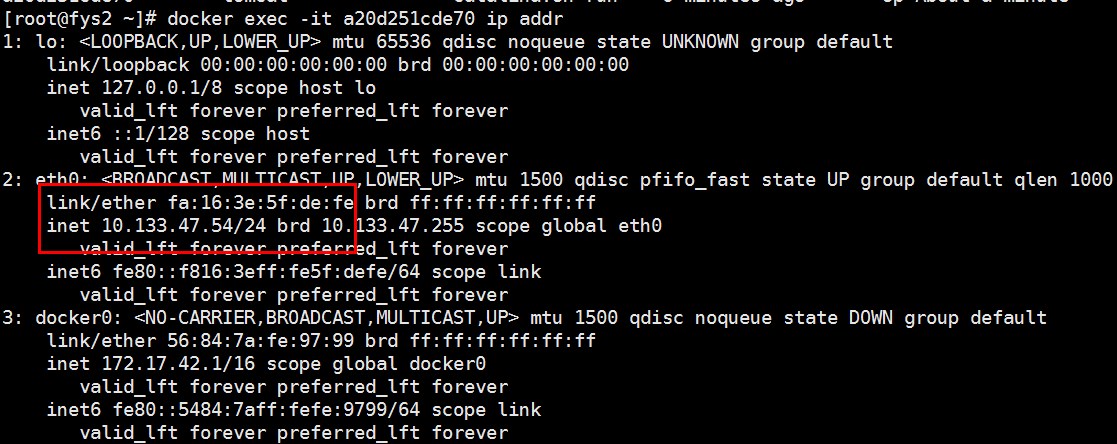
*[root@fys2 ~]# netstat -anp|grep 8080*

*tcp 0 0 :::8080 :::\* LISTEN 23725/java*

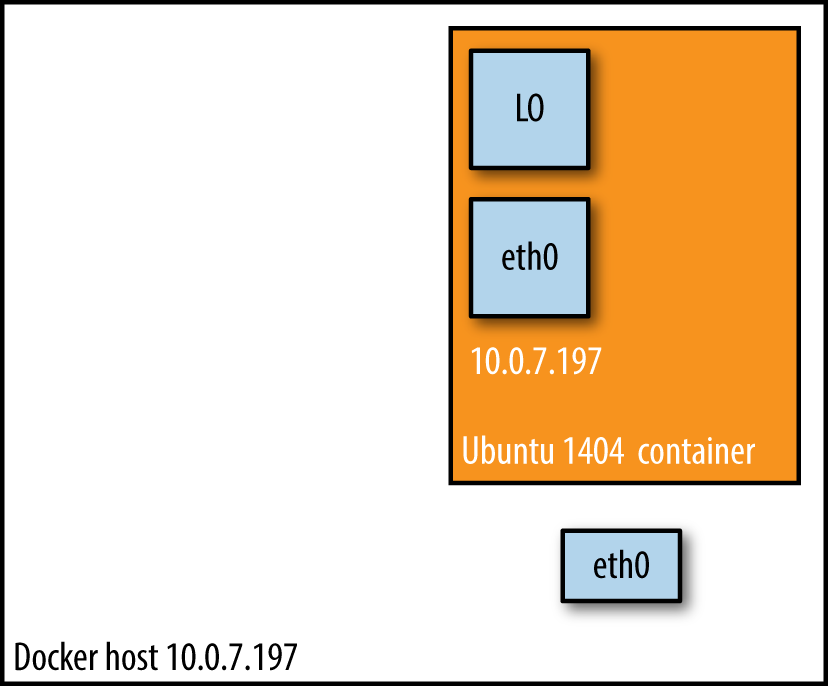


直接访问主机的8080端口

通过下面的指令查询使用的ip地址，如下：

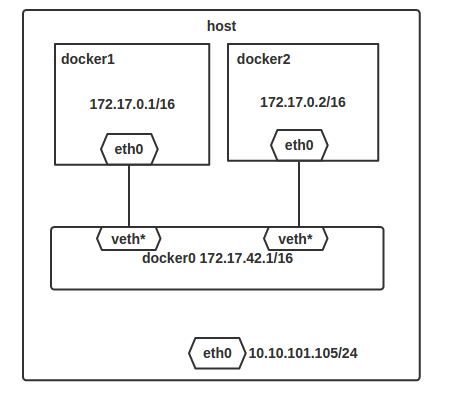


从该例子上可以看出，容器和宿主机使用相同的IP地址，下图是使用host模式网络时，容器实际上继承了宿主机的IP地址，该模式比bridge模块更快，但是直接将容器暴露在公共网络中，具有安全隐患：



# 2.bridge模式网络

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，下图是bridge模式的拓扑：



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。

接下来就要为容器分配IP了，Docker会从RFC1918所定义的私有IP网段中，选择一个和宿主机不同的IP地址和子网分配给Docker0，连接到docker0的容器就从这个子网中选择一个未占用的IP使用。如一般Docker会使用172.17.0.0/16这个网段，并将172.17.42.1/16分配给docker0网桥（在主机上使用ifconfig命令是可以看到docker0的，可以认为它是网桥的管理接口，在宿主机上作为一个虚拟网卡使用）。

Docker完成以上网络配置的过程大概是这样的：

1）在主机上创建一个虚拟网卡veth pair设备，veth设备总是成对出现，它们祖恒了一个数据通道，数据从一个设备进入，就会从另一个设备处理，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，如下所示：

*[root@fys2 ~]# brctl show*

*bridge name bridge id STP enabled interfaces*

*docker0 8000.56847afe9799 no*

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

下面介绍bridge模式是如何通信的：

在bridge模式下，连同在同一网桥上的容器可以互相通信（若处于安全考虑，可以禁止他们之间通信，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，主机上的iptable规则，命令为iptables -vnL，可以看到：

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

这条规则将源地址为172.17.0.0/16的包（也就是从Docker容器产生的包），并且不是Docker网卡发出的，进行源地址转换，转换成主机网卡的地址，举个例子，假设主机有一块网卡为eth0，IP地址为10.10.101.105/24，网卡为10.10.101.254。从主机上一个IP为172.17.0.1/16的容器中ping百度。IP包首先从容器发往自己的默认网关docker0，包到达docker0以后，也就是达到了主机，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，接着包转发给eth0，并从eth0发出去。在外界看来，这个包是从10.10.101.105上发出来的，Docker对外是不可见的。

外部的机器是如何访问Docker容器的服务？通过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 8081:8080 tomcat*

查看iptable的规则变化，发现多出一条：

*tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.1 tcp dpt:8080*

此条规则就是对主机eth0收到的目的端口为8081的tcp流量进行DAT转换，将流量发往172.17.0.1:8080，也就是上面创建的Docker容器，所以外界只需要访问10.10.101.105:80就可以访问到容器中的服务。除此之外，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，但是工作方式还是一样的。

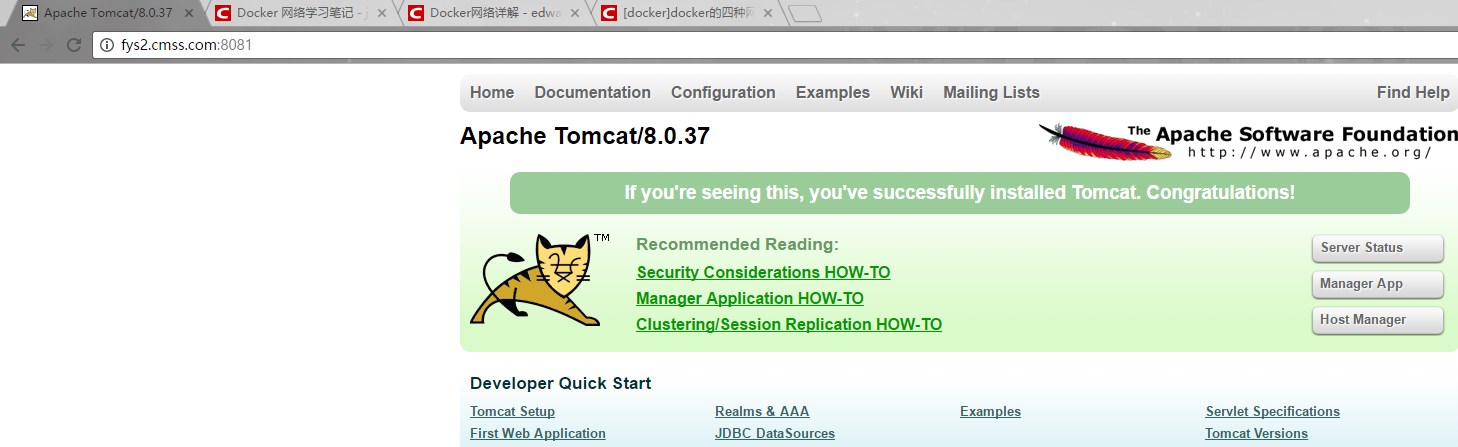
*[root@fys2 ~]# docker run -d --name web -p 8081:8080 tomcat*

*[root@fys2 ~]# docker ps*

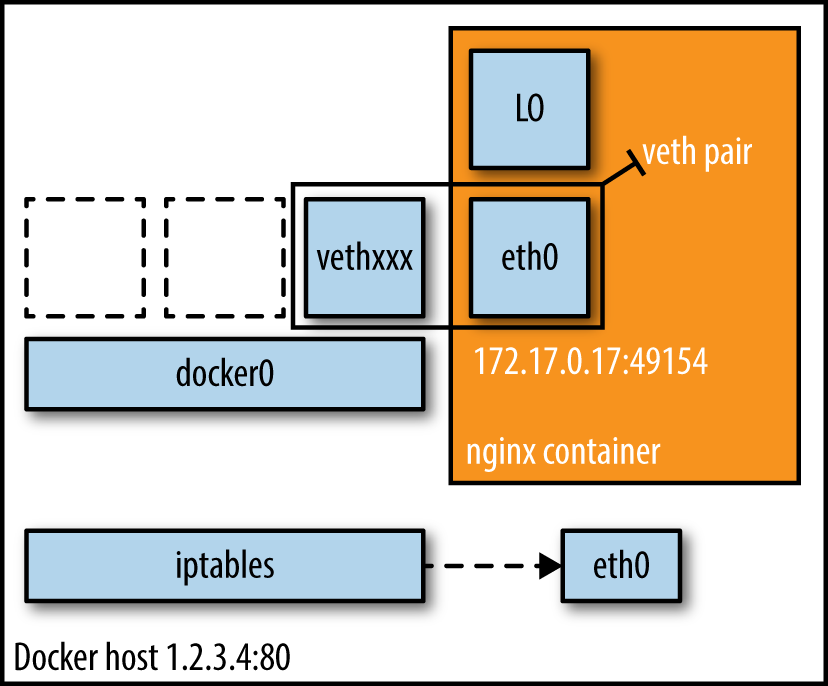
*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

*807ac83e6a44 tomcat "catalina.sh run" 36 minutes ago Up 36 minutes 0.0.0.0:8081->8080/tcp web*

访问页面如下：



访问模式如下：



# 3 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Kubernets使用的网络模式

*[root@fys2 ~]# docker run -d -P --net=bridge nginx:1.9.1*

*[root@fys2 ~]# docker ps*

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

963bbd363db3 nginx:1.9.1 "nginx -g 'daemon of About a minute ago Up About a minute 0.0.0.0:32769->80/tcp, 0.0.0.0:32768->443/tcp sick\_pare

*[root@fys2 ~]# docker exec -it sick\_pare 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

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

76: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::42:acff:fe11:2/64 scope link

valid\_lft forever preferred\_lft forever

*[root@fys2 ~]# docker run -it --net=container:sick\_pare ubuntu:14.04 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

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

76: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::42:acff:fe11:2/64 scope link

valid\_lft forever preferred\_lft forever

结果上显示，第二个容器使用了--net=container的参数，因此与第一个容器具有相同的IP地址

# 4 none模式网络

在该模式下，容器在宿主的网络栈中，但是不进行任何配置，实际上该模式关闭了容器的网络功能，在下面两种情况下：

容器不需要网络

用户自己定义网络

使用这种模式，示例如下：

*[root@fys2 ~]# docker run -d -P --net=none nginx:1.9.1*

7810fec9b0610a501324110aba4fabb2d1b7f5d2840ded34a51b60224406db18

*[root@fys2 ~]# docker ps*

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

7810fec9b061 nginx:1.9.1 "nginx -g 'daemon of 3 seconds ago Up 2 seconds ecstatic\_turing

*[root@fys2 ~]# docker inspect 7810fec9b061|grep IPAddress*

"IPAddress": "",

"SecondaryIPAddresses": null,

# 5 docker网络基础配置

大量的互联网应用服务包括多个服务组件，这往往需要多个容器之间通过网络通信进行互相通信。在启动容器的时候，如果不指定对应参数，在容器外部是无法通过网络来访问容器内的网络应用和服务的。Docker目前提供映射容器端口到宿主主机和容器互联机制来为容器提供网络服务。

## 5.1 端口映射实现访问容器

在容器中运行一些网络应用，要让外部访问这些应用时，可以通过-P或者-p参数来指定端口映射。当使用-P会随机映射一个随机的端口至容器内部开发的网络端口:

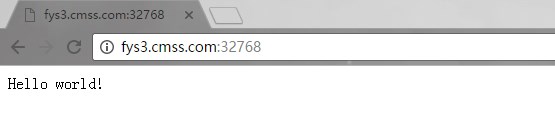
[root@fys3 docker-tomcat]# docker run -d -P training/webapp python app.py

[root@fys3 docker-tomcat]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

bb0945ef821a training/webapp "python app.py" 54 seconds ago Up 53 seconds 0.0.0.0:32768->5000/tcp grave\_bohr

可以看到本地主机的32768被映射到容器的5000端口，访问主机的32768即可访问容器内Web容器提供的页面



## 5.2 容器互联实现容器间的通信

容器的连接（linking）系统是除了端口映射外另一种可以与容器中应用进行交互的方式。它会在源和接收容器见创建一个隧道，接收容器可以看到源容器指定的信息。

1) 自定义容器命令

Linking系统依据容器的名称来执行，因此需要自定义一个好记的容器命名（系统默认会分配一个名字），但是自定义命名好记，当要连接其他容器时候，可以作为一个有用的参考点，比如连接Web容器到db容器。使用--name来为容器自定义命令

*[root@fys3 docker-tomcat]# docker run -d -P --name web training/webapp python app.py*

*[root@fys3 docker-tomcat]# docker ps*

b46f4d0c4423 training/webapp "python app.py" 4 seconds ago Up 4 seconds 0.0.0.0:32769->5000/tcp web

*[root@fys3 docker-tomcat]# docker inspect -f "{{.Name}}" b46f4d0c4423*

/web

2）容器互联

使用--link参数可以让容器之间安全的进行交互，下面创建一个数据库容器：

*$docker run -d --name db training/postgres*

然后创建一个web容器，并将它连接到db容器

*$docker run -d -P --name web --link db:db training/webapp python app.py*

此时，db容器和web容器之间建立互联关系，使用docker ps 可以看到容器见的连

参考文献

http://www.infoq.com/cn/articles/docker-network-and-pipework-open-source-explanation-practice

http://www.cnblogs.com/leetieniu2014/p/5465164.html