Docker(三) 网络模式

选择网络模式

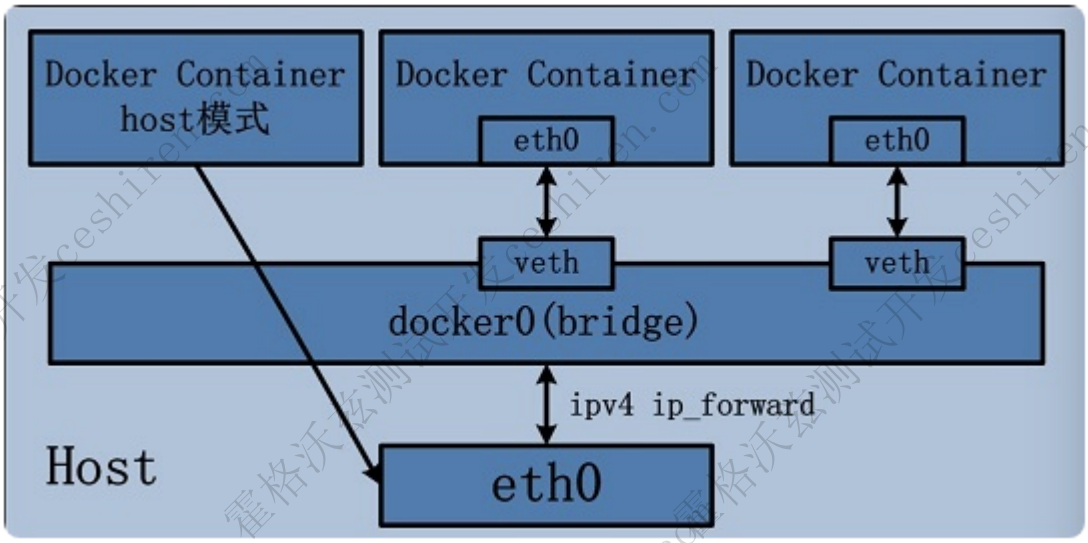
在使用 docker run 命令的时候 跟上 net 或者 network 参数就可以选择使用的网络模式了， 默认是 briage 模式，其他的还有 container，host 和 none。

briage 模式

其实我们上一篇帖子中介绍的就是 briage 模式的原理，docker 会创建一个名字叫 docker0 的 linux 网桥来连通所有的容器。如果外网想要访问容器就需要使用端口映射。 这是 docker 默认的玩法，我们在上篇帖子中已经很详细的介绍了，这里就不多做阐述了。

host 模式

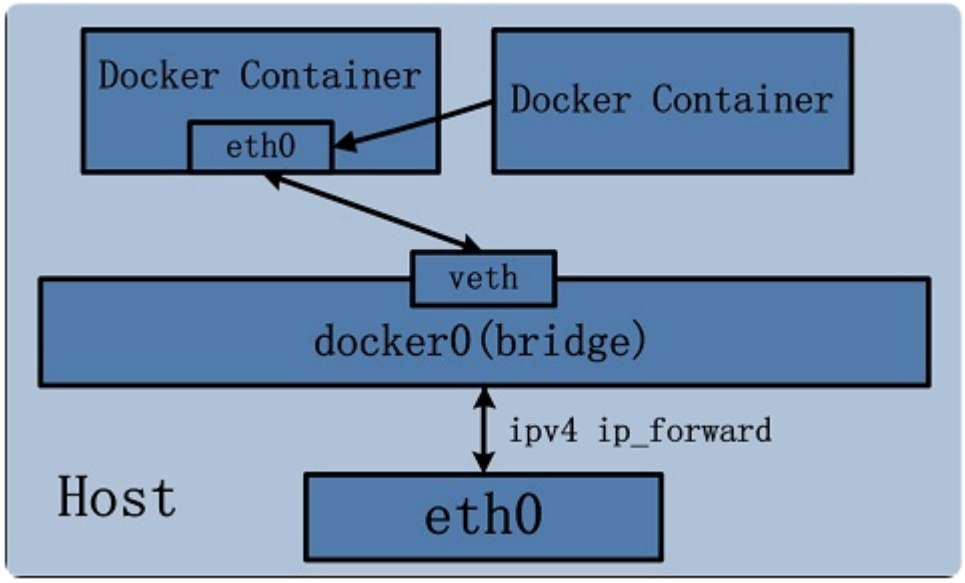
briage 模式是的作用是创建一个小型的私有局域网。 但有些时候我们希望容器能够拥有宿主机的网络环境。 所以 host 模式就这是么诞生的。



上面就是一个 host 模式和 briage 模式的对比图。 看过上一篇帖子的小伙伴应该知道 briage 模式使用虚拟网卡设备对和 docker0 网桥来构建私有网络。 而 host 模式不是的，它直接使用宿主机的网卡，这样容器就有了跟宿主机一样的网络环境。 我们进入容器后使用 ifconfig 命令看到的 ip 也是跟宿主机一样的。 通常我们的编译容器就是用 host 模式启动的，因为 host 模式的优点就是不必再特意配置网络环境， 而我们的编译容器通常都需要拉取代码，配置权限等操作。所以只要宿主机的网络和权限是 OK 的，那么编译容器就是 OK 的。 同样的我们某些测试服务也是可以使用 host 模式启动的，最大的好处当然就是不需要使用端口映射了。

container 模式

这是一个重要的模式，它的作用是可以把 N 个容器的网络环境绑定在一个容器上，这么说有点拗口。 直接看下面的原理图。



就是说，我们先启动一个容器 (假如是用 briage 模式启动). 然后启动其他容器的时候在 net 或者 network 参数的设置上使用container:容器名称。 这样后启动的容器都会使用第一个容器的网卡。 也就是说，虽然我们启动的是不同的容器，但他们彼此之前是可以使用 localhost 进行通讯的。这个模式非常有用。 想象下面一个场景， 通常我们要部署一个产品的时候都比较希望一个模块一个容器进行部署，这样比较方便管理。当更新某一个服务的时候完全去替换目标模块的容器就可以了。 这样比把所有的服务都安装在一个容器中好管理的多。 并且在之后的 k8s 中的文章中也会提到，
在做高可用和负载均衡策略中也是要把服务单独拆成一个容器来做的，这符合微服务的设计原则，事实上 k8s 的 pod 就是用 container 模式启动的，里面所有的容器都使用 pod 中默认的 pause 容器的网络。 所以如果要实现这种目的用 briage 模式或者 host 模式就会比较尴尬。 一般使用 host 模式比较好的场景是以下两点：

- 容器执行一些一次性任务，并不启动进程进行持久的服务。这样就没有端口冲突问题。
- 确保启动的每种服务类型只有一个实例， 如果启动多个相同的服务 (多套测试环境更或者负载均衡场景) 就需要端口映射，这样比较麻烦。

如上所说，如果我们是以部署多套测试环境为目标的话，用 host 模式会导致端口冲突。 那么如果用 briage 模式呢？ 当然 briage 模式下的每个容器都是一个独立的网络环境，所以并不存在端口冲突问题。 但它也带来了配置管理问题，无法用 localhost 访问目标服务的话就需要知道其他所有服务的 ip 地址。而在 briage 模式下容器启动前是无法知道 ip 地址的，它们的 ip 都是由 docker 随机分配。 之后重新启动容器的时候 ip 还会变化。 所以这时候使用 braige + container 模式就能解决这个问题。 一个 docker 宿主机中的多套测试环境下，只要每套环境的第一个容器是使用 braige 模式启动的就可以了，他们就达到了网络隔离的目的。 然后其余的容器分别使用 container 模式绑定到这个容器上。
就可以避免端口冲突的同时，解决配置管理问题，因为一套环境下的容器都可以用 localhost 互相访问了。

none 模式

这同样是一个大量使用的模式，它把设置容器网络环境的责任交给了用户自己。 使用 none 模式启动的容器没有任何的网络设置，是完全的无网络状态。 我们在使用 none 模式启动容器后可以使用 pipework 这种第三方工具设置它的网络环境。这是一种重要的使用方式。实际上这是单机的 docker 网络最常用也是最好的解决方案。 上面我们说的 briage + container 模式虽然同时解决了端口冲突和容器互联的问题。 但是它还是有一个缺点， 那就是第一个容器是使用 briage 模式启动的，而 briage 模式启动的容器外界是无法访问的，想访问就必须做端口映射。
那么如果我们有很多套环境，每套环境中都有很多服务需要外界访问的话。 维护一个庞大的端口映射列表是不可避免的。
而如果我们能有一种方法，能让容器的网络和公司的内网打通的话。就可以解决这个问题。 下面我们讲一下这个做法的原理。

网络基础

首先普及一下计算机网络的一个基础知识。说一下为什么我们默认是访问不了容器的网络的。 因为 docker 为容器建立的是一个私有网络，里面的 ip 地址也都是私有 ip 地址。 与我们公司内网中使用的不是一个网段。 假设我们 ping 一台机器，这时候我们的本机首先会把目标的 ip 地址与本机的子网掩码做与操作。
如果发现是同一个网络的话就会发送到交换机做广播。如果发现是不同网络的 ip 地址就去路由器根据路由表进行转发。
我们的路由器都有自己学习的功能，如果它收到的请求不在路由表里，它就会发广播包去寻址，等待目标机器回应然后记录在路由表里。
但这里有一个规则，就是它不记录也不转发私有 ip。 什么是私有 ip 呢？ 就是我们使用的 A,B,C 类 ip 地址中都预留了一个 IP 地址段来给各个组织搭建自己的局域网络。 而 docker 为所有容器分配的 ip 地址就是私有 ip。 所以容器的 ip 是绝对不会进入路由的，我们是也是无法直接访问容器的网络。 根据这个原理，只要让我们的容器的 ip 地址能进入路由表中或者说我们干脆能给容器分配一个跟宿主机同样网段的 ip 地址， 就可以解决这个问题。

linux briage

这里我们又提到了网桥， 因为要实现上面说的目的需要利用 linux 的网桥来实现。 上篇帖子曾经讲过，linux 网桥有个特点是它既能支持 2 层协议，也可以支持 3 层协议。当我们给一个 linux 网桥分配一个 ip 地址的时候，就等于是开启了网桥的 3 层功能，也就是说这个时候你可以把网桥当成一个带有路由功能的 3 层交换机。 所以我们要做的事情有以下几步。

- 在宿主机上创建一个 linux 网桥，br0.
- 把宿主机的网卡连接到这个网桥上并把宿主机的 ip 设置给 br0. 这样 br0 就拥有了宿主机的 ip 地址，同时原来宿主机的网卡就没有 ip 了，但是它是挂在 br0 上的。所以我们使用 br0 的 ip 也是可以访问宿主机的 (网桥会转发)。
- 修改 docker 的启动参数。将 DNS，ip 网段，网关等信息都设置成与公司内网相匹配的值，网桥从 docker0 换成 br0。
  注意容器的网段一定要跟宿主机一致
- 启动容器的时候使用 none 模式，然后用 pipework 分配与宿主机相同网段的 ip。 或者使用 briage 模式启动，让它自动分配 ip。

通过这种方式我们就可以直接使用 ip 地址访问容器网络了。 当然这里推荐使用 none 模式 + container 的方式代替 braige + container。因为使用 pipework 是可以由我们自己设置固定 ip 地址的。

none 模式玩法的具体步骤

下面是我之前搞的一个小文档的内容， 在 centos6.5 上搞的。

安装 pipework

```
yum install -y net-tools
yum install -y bridge-utils
yum -y install git
git clone https://github.com/jpetazzo/pipework
cp pipework/pipework /usr/local/bin/
```

网络环境配置

我本机的宿主机网卡名称为 ens33,
为了保证网络能够正常使用，我这边做法先把配置文件拷贝到单独文件夹进行编写；

```
[root@localhost docker]# cd /opt/
[root@localhost docker]# mkdir docker
[root@localhost opt]# cd docker/
[root@localhost docker]# mkdir networkBak
[root@localhost docker]# cd networkBak/
[root@localhost networkBak]# mkdir bak
[root@localhost networkBak]# cp /etc/sysconfig/network-scripts/ifcfg-ens33 ./
[root@localhost networkBak]# ls bak ifcfg-ens33
[root@localhost networkBak]# cp ifcfg-ens33 ./bak/
[root@localhost networkBak]# cp ifcfg-ens33 ifcfg-br0
```

vi ifcfg-ens33，修改成以下信息

```
[root@localhost networkBak]# cat ifcfg-ens33
TYPE="Ethernet"
NAME="ens33"
UUID="5ef41f83-92a7-429d-a83a-6444acd2545e"
DEVICE="ens33"
ONBOOT="yes"
BRIDGE="br0"
```

vi ifcfg-br0,修改成以下信息，名称记得一定要改成 br0

```
[root@localhost sysconfig]# cat network-scripts/ifcfg-br0
TYPE="Bridge"
BOOTPROTO=static
NAME="br0"
DEVICE="br0"
ONBOOT="yes"
IPADDR=192.168.11.94
NETMASK=255.255.254.0
GATEWAY=192.168.11.1
DNS1=192.168.11.1
DNS2=8.8.8.8
DNS3=114.114.114.114
```

把这两个文件替换到 /etc/sysconfig/network-scripts/下

```
[root@localhost networkBak]# ls bak ifcfg-br0 ifcfg-ens33
[root@localhost networkBak]# cp ifcfg-br0 ifcfg-ens33 /etc/sysconfig/network-scripts/
cp"/etc/sysconfig/network-scripts/ifcfg-ens33" y

[root@localhost networkBak]# service network restart
```

三、修改 docker0 默认的网络地址信息
vi 修改 docker-network 如下
"-b br0 --iptables=false --dns=192.168.11.1 --dns=8.8.8.8 --dns=114.114.114.114 --fixed-cidr=192.168.11.100/23"
PS: --fixed-cidr 必不可少，必须与宿主机的 IP 在同一网段，表示的含义是该 IP 将作为容器的起始地址。 然后重启 docker