2021.3.13 Docker 网络模型详解

一．Docker 网络基础

Docker 目前对单节点设备提供了将容器端口映射到宿主机和容器互联两个网络服务。

1 端口映射

在 Docker 中容器默认是无法与外部通信的，需要在启动命令中加入对应的参数才允许容器与外界通信。

当 Docker 中运行一个 Web 服务时，需要把容器内的 Web 服务应用程序端口映射到本地宿主机的端口。这样，用户访问宿主机指定的端口的话,就相当于访问容器内部的 Web 服务端口。

1）使用-P 选项时 docker 会随机映射一个端口至容器内部开放

```
[root@crushlinux ~]# docker run -d -P --name test1 nginx
```

使用 docker port 可以查看端口映射情况（后面跟上容器名）

```
[root@crushlinux ~]# docker port  test1
```

网上下载的源码包，查看输出日志

```
[root@crushlinux ~]# docker logs test1
```

制作的源码包的日志位置（需要进入到容器内查看）

```
[root@crushlinux ~]# docker exec -it abc /bin/bash    #进入到容器内

[root@crushlinux ~]# cat /usr/local/nginx/logs/access.log  #查看日志内容
```

2）--p 的三种方式：

第一种

使用-p 可以指定映射到本地所有 IP 的 80 端口

```
[root@crushlinux ~]# docker -d -p 8000:80 --name test1 nginx
[root@crushlinux ~]# docker port  test1
```

第二种

映射到指定 IP 的指定端口

```
[root@crushlinux~]# docker run -d -p 192.168.200.12:8001:80 --name test2 nginx
[root@crushlinux ~]# docker port test2
```

注意事项：

第一种的含义代表为映射主机上所有 IP 的 80 端口

第二种的含义代表为映射指定 IP 的 80 端口（具有绝对性）

第三种

映射到指定 IP，但是宿主机端口的随机分配的

```
[root@crushlinux~]#  docker run -d -p 192.168.200.12::80 --name test3 nginx
[root@crushlinux ~]# docker port  test3
```

·指定传输协议

```
[root@crushlinux~]# docker  run -d -p 80:80/tcp --name test5 nginx
[root@crushlinux ~]# docker port  test5
```

2.端口暴露

咱们之前讲过 EXPOSE 命令用于端口暴露,很多人会把端口暴露和端口映射混为一谈 3 目前有两种方式用于端口暴露，--expose 和 EXPOSE 方式,这两种方式作用相同,但是--expose 可以接受端口范围作为参数，例如--expose=2000~3000。

Dockerfile.的作者一般在包含 EXPOSE 规则时都只提示哪个端口提供哪个服务。访问时还需要运维人员通过端口映射来指定。--expose 和 EXPOSE 只是为其他命

令提供所需信息的元数据。

通过 docker inspect container name：查看网络配置：

```
[root@docker ~]# docker inspect test1
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"c83ae4368c468d3054eede1a83a3a67fe9dc375013f4a585387e5e16243406b3",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "32768"
                    }
                ]
            },
```

3. 容器互联（作为了解）

容器互联是除了端口映射外另一种可以与容器通信的方式。端口映射的用途是宿主机网络与容器的通信，而容器互联是容器之间的通信。

当前实现容器互联有两种方式，一种是把两个容器放进一个用户自定义的网络中，另一种是使用--link 参数（已经弃用，即将删除的功能）。

为什么要使用一个单独的网络来连接两个容器呢？设想一下后端容器需要调用一个数据库环境，数据库容器和后端服务容器如果使用上下文中的暴露端口或者映射端口来通信，势必会把数据库的端口也暴露在外网中，导致数据库容器的安全性大大降低，为了解决这个问题，docker 允许用户建立一个独立的网络来放置相应的容器，只有在该网络中的容器才能相互通信，外部容器是无法进入这个特定网络中的。

一个容器可以同时加入多个网络，使用不同地址可以访问不同网络中的容器。

用户自定义的网络

首先创建两个容器，命名为 container1 和 container2

```
[root@docker ~]# docker run -itd --name=container1 busybox
6200098cb616e6f414b39934153bb18388908547cd1049ea94851df57ec4b74f
[root@docker ~]# docker run -itd --name=container2 busybox
2d77db9b4415f9446f700cd7fb4ad7cf5352797bf79dd712dc69a5ebb139a1f7
```

接下来创建一个独立的容器网络，这里使用 bridge 驱动（桥接模式），其他可选的值还有 overlay 和 macvlan。

```
[root@docker ~]# docker network create -d bridge --subnet 172.25.0.0/16 demo_net
2abcbbf2ecae053b2abdf04fe3169359ca1720711326010762f8fa5e7a08abc5
[root@docker ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
a6f1f7907e11        bridge              bridge              local
2abcbbf2ecae        demo_net            bridge              local
e963d0ca867a        host                host                local
3995cdd212c7        none                null                local
```

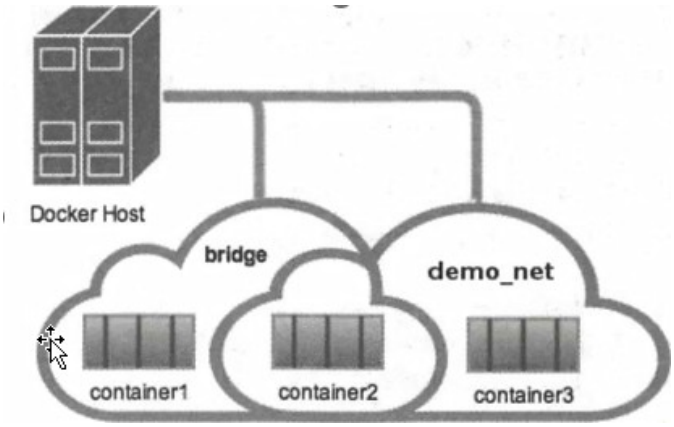
使用--subnet 和--gateway 可以指定子网和网关，现在我们把 container2 加入到 demo_net 中

```
[root@docker ~]# docker network connect demo_net container2
[root@docker ~]# docker network inspect demo_net
        "Containers": {
            "2d77db9b4415f9446f700cd7fb4ad7cf5352797bf79dd712dc69a5ebb139a1f7": {
                "Name": "container2",
                "EndpointID":
"edccd77eb0fc4a1bf928109ebe4a832a51b506ede59455d934184c1ae26439b5",
                "MacAddress": "02:42:ac:19:00:02",
                "IPv4Address": "172.25.0.2/16",
                "IPv6Address": ""
            }
        },
```

使用 docker network inspect 可以查看网络中容器的连接状态。Container2 已经在 demo_net 网络中，注意 IP 地址使自动分配的。

启动第三个容器：

```
[root@docker ~]# docker run --network=demo_net --ip=172.25.3.3 -itd --name=container3
busybox
e480fceffe8be7bbc3bed1c2adca16f75d121a1832dabd6e91a80c7ee6969148
```



查看三个容器内部的网络

```
[root@docker ~]# docker exec -it container1 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:31 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
```

Docker 网络模式

安装 docker 时会自动创建三个网络，可以使用 docker network ls 列出这些网络

```
[root@crushlinux ~]# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
581741560f8d   bridge    bridge    local
3cf86a23f021   host      host      local
013a6c6040e2   none      null      local
```

我们在使用 docker run 创建容器时，可以用--net 选项来指定容器的网络模式

Docker 有以下四种模式

·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

·四种模式的介绍

1. host 模式

Docker 底层使用了 Linux 的 Namespaces 技术来进行资源隔离，如 PID

Namespace 隔离进程，Mount Namespace 隔离文件系统，Network Namespace 隔离网络等。一个 NetworkNamespace 提供了一份独立的网络环境,包括网卡、路由、iptable 规则等都与其他的 NetworkNamespace 隔离。一个 Docker"容器一般会分配一个独立的 Network Namespace。但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 Root 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。出于安全考虑不推荐使用这种网络模式。

我们在 192.168.200.111/24 的机器上用 Host 模式启动一个含有 WEB 应用的 Docker 容器，监听 TCP 80 端口。当我们在容器中执行任何类似 ifconfig 命令查看网络环境时，看到的都是宿主机上的信息。而外界访问容器中的应用，则直接使用 192.168.200.111:80 即可，不用任何 NAT 转换，就如直接跑在宿主机中一样。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。

指定一个容器的网络模式

```
[root@crushlinux ~]# docker run -itd --net=host --name=host busybox
```

在查看容器的网络模式：

```
[root@crushlinux ~]# docker exec -it host ifconfig
```

在和宿主机执行 ifconfig 命令

```
[root@crushlinux ~]# ifconfig
```

两者对比基本相同

2. container 模式

这个模式可以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。

使用--net=containex:容器 id/容器名称 name,多个容器使用共同的网络看到的 ip 是一样的。

示例：指定一个容器的网络模式

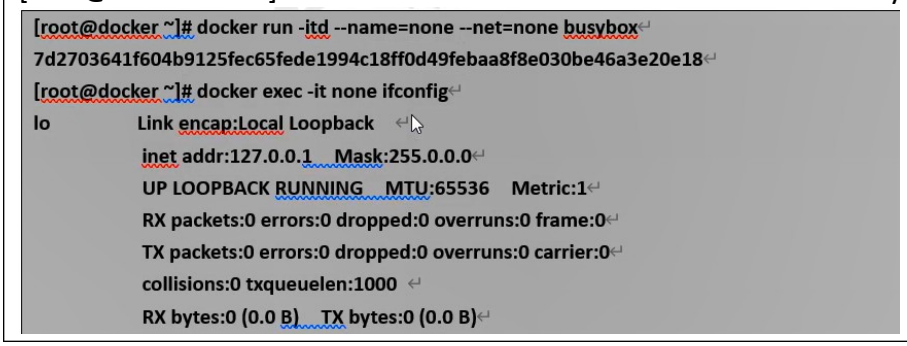
```
[root@crushlinux ~]# docker run -itd  --net=container:con1 --name=con2 busybox
```

3.none 模式

在这种模式下，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 I 等。

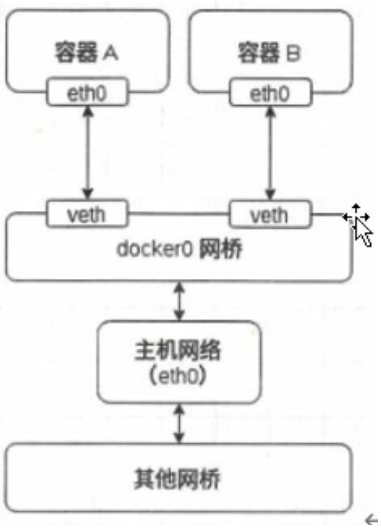
使用--net=none 指定，这种模式下不会配置任何网络。

```
[root@crushlinux ~]# docker  run -itd  --name-none --net=none busybox
```



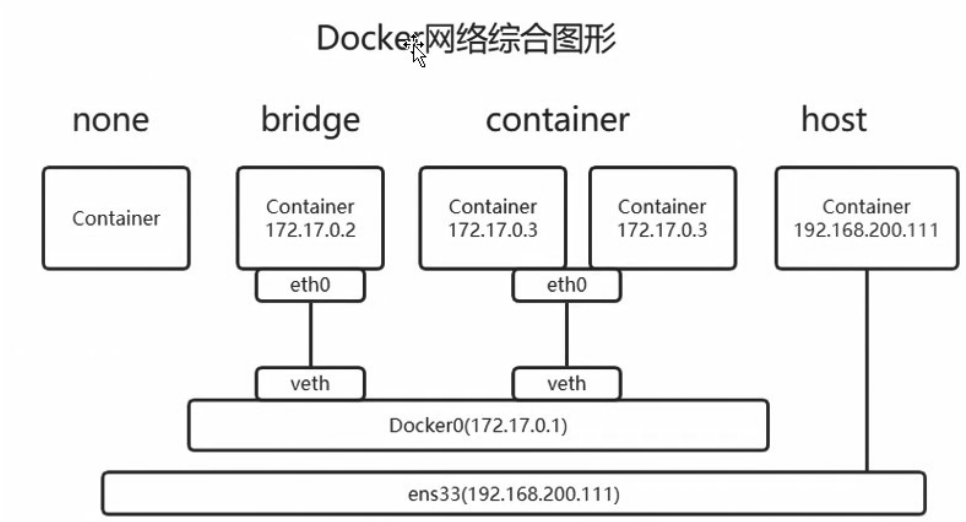
4.bridge 模式

bridge 模式是 Docker 默认的网络设置，属于一种 NAT 网络模型，Docker daemon 在启动的时候就会建立一个 dockerO 网桥（通过-b 参数可以指定)，每个容器使用 bridge 模式启动时，Docker 都会为容器创建一对虚拟网络接口（veth pair）设备，这对接口一端在容器的 Network Namespace，另一端在 docker0，这样就实现了容器与宿主机之间的通信。



在 bridge 模式下，Docker 容器与外部网络通信都是通过 iptables 规则控制的，这也是 Docker 网络性能低下的一个重要原因。使用 iptables -wnl -t nat 可以查看 NAT 表，在 ChainDocker 中可以看到容器桥接的规则。

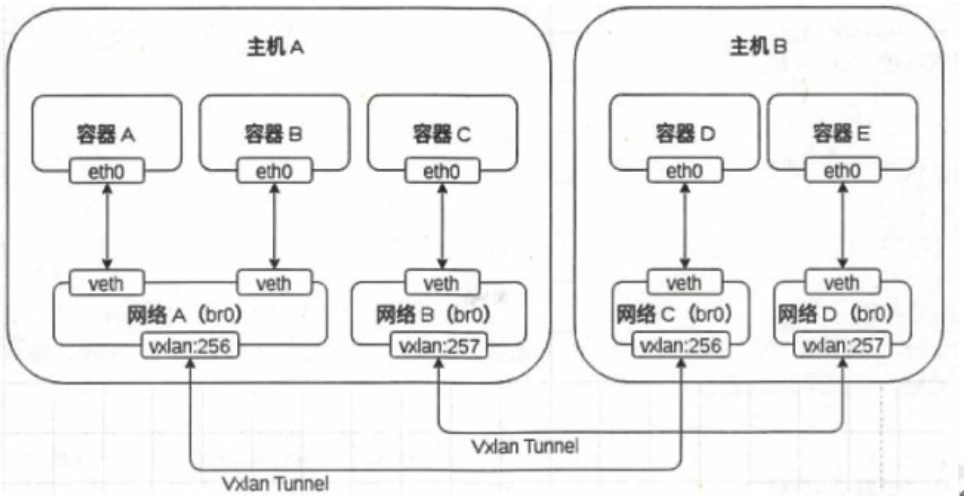
```
[root@crushlinux ~]# iptables  -vnl -t nat
```



5.overlay 模式（了解）

这是 Docker 原生的跨主机多子网的网络模型，当创建一个新的网络时，Docker 会在主机上创建一个 Network Namespace，Network Namespace 内有一个网桥，网桥上有一个 vxlan 接口，每个网络占用一个 vxlanID，当容器被添加到网络中时，Docker 会分配一对 veth.网卡设备，与 bridge 模式类似，一端在容器里面，另一端在本地的 Network Namespace 中。

容器 A、B、C 都在主机 A 上面，而容器 D、E 则在主机 B 上面，现在通过 Overlay 网络模型可以实现容器 A、B、D 处于同一个子网，而容器 C、E 则处于另一个子网中。



Overlay 中有一个 uxlan ID，值得范围为 256~1000，vxlan,隧道会把每一个 ID 相同的网络沙盒连接起来实现一个子网。