Docker 网络基础

本篇是第七部分“网络篇”的第一篇。在这个部分，我会为你由浅入深的介绍 Docker 网络相关的内容。包括 Docker 网络基础及其实现和内部原理等。本篇，我先来为你介绍 Docker 网络基础。

Docker 网络

之前的内容中，我已经为你介绍过很多使用 Docker 来管理容器，或是增强容器安全等内容。在本篇中，我将为你介绍 Docker 网络相关的内容。

在《深入剖析容器》 和 《容器的核心：Namespace》 这两篇中，我为你介绍过 Docker 容器使用的一项核心技术，便是 Linux namespace。

Linux namespace 有很多类，与 Docker 容器网络联系最为紧密的便是 Network namespace 了。使用它可以用来控制网络堆栈，开放端口等能力。

这里我们先来看看 Docker 中默认的 network 资源，并对其分别进行介绍。

复制

```
(MoeLove) ➜  ~ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
319ea7ae2a9e        bridge              bridge              local
e07891351b5e        host                host                local
46716de300ca        none                null                local
```

可以看到默认情况下有三大类。

另外，在启动容器的时候，可以通过 `--network` 参数进行指定容器所使用的 network 的名称。

none 网络

顾名思义，none 网络使用了 null 驱动，是没有网络的意思。

复制

```
(MoeLove) ➜  ~ docker run --rm --network none -it alpine sh
/ # ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # ping -c 1 moelove.info
ping: bad address 'moelove.info'
/ # ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1): 56 data bytes
ping: sendto: Network unreachable
```

可以看到，该容器只有 lo 接口，且访问不到外部网络。

host 网络

host 网络，是一种特殊的网络模式，当使用了此网络模式，容器将与主机共享网络堆栈。我们来看看容器使用此网络模式时的情况。

复制

```
(MoeLove) ➜  ~ docker run --rm --network host -it alpine sh
/ # ping -q -c 1 moelove.info
PING moelove.info (185.199.109.153): 56 data bytes

--- moelove.info ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 83.418/83.418/83.418 ms
```

可以看到它是可以访问外部网络的，同时如果你去检查 /etc/hosts 文件，你会发现内容与主机上是相同的。

我们来具体看看此容器的 Network namespace 的情况：

复制

```
(MoeLove) ➜  ~ sudo ls --time-style="+" -l  /proc/`docker inspect $(docker ps -ql
lrwxrwxrwx. 1 root root 0  net -> net:[4026531992]
```

与主机上任意进程的 Network namespace 进行对比：

复制

```
(MoeLove) ➜  ~ ls --time-style="+" -l /proc/self/ns |grep net
lrwxrwxrwx. 1 tao tao 0  net -> net:[4026531992]
```

可以看到它们使用的 Network namespace 是相同的。

bridge 网络

bridge 网络是最常用，也是默认的网络模式。使用此网络模式，容器会在启动时，由 Docker 为其分配一个 IP 地址，并且访问外部网络的时，会通过 Docker 默认提供的 docker0 接口出去。

复制

```
(MoeLove) ➜  ~ docker run --rm --network bridge -it alpine sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:xxxxx:00:03
          inet addr:172.17.0.3  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3572 (3.4 KiB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # ip r
default via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 scope link  src 172.17.0.3
```

查看主机上 docker0 接口的信息：

复制

```
(MoeLove) ➜  ~ ifconfig docker0
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 xxxxxx:be8  prefixlen 64  scopeid 0x20<link>
        ether 02xxxxxx:e8  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 262  bytes 44560 (43.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

由此可以看出，所谓的 bridge 便是使用 docker0 作为容器网络的 bridge。

container 网络

这里我来为你介绍另外一种使用 Docker 网络的方式，即 container 模式。所谓的 container 网络是指为容器共享指定容器的网络堆栈。

我们来实际看看它的使用方式：

复制

```
(MoeLove) ➜  ~ docker run --rm -d redis:alpine
634752234ad169710a883f615e487508b278b910a0e00a0bedf193a7c17828e8
(MoeLove) ➜  ~ docker run --rm -it --network container:$(docker ps -ql) alpine sh
/ # ps -ef
PID   USER     TIME  COMMAND
    1 root      0:00 sh
    6 root      0:00 ps -ef
/ # netstat  -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PI
tcp        0      0 0.0.0.0:6379            0.0.0.0:*               LISTEN      -
tcp        0      0 :::6379                 :::*                    LISTEN      -
```

通过指定 `--network container:$容器ID` 便可使用指定容器网络。

从上面例子的输出中可以看到，虽然新启动的容器中并未运行任何监听 6379 端口的进程，但却能查看到当前 6379 端口已经被监听。（这是被开始启动的 Redis 容器所监听的）

这个共享网络堆栈的方法，后来也被用到了 Kubernetes 的 POD 网络中了。

其他的网络驱动

其实 Docker 除了上述几种外， Docker 默认还支持其他的网络驱动，可以通过以下命令查看：

复制

```
(MoeLove) ➜  ~ docker info --format "{{ .Plugins.Network }}"
[bridge host ipvlan macvlan null overlay]
```

可以看到还有其他包括 ipvlan、macvlan、overlay 等网络驱动可用。不过通常情况下，这些模式是配合具体的场景和需求来使用的，本篇我们暂且跳过。

总结

本篇，我为你介绍了 Docker 的几种容器网络模式的使用，即 none、host、bridge 和 container 等模式。

这四种模式，基本满足了全部 Docker 容器网络的使用需求，在实际应用中，可按需求灵活使用。

下一篇，我们将重点放在 bridge 网络上，学习如何“定制 bridge”。