定制 bridge

本篇是第七部分“网络篇”的第二篇。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。上篇，我已经为你介绍了 Docker 网络基础。本篇，我们将重点放在 bridge 网络上。

在上篇《Docker 网络基础》中，我们已经知道 Docker 常见的几种网络模式。其中最为常用的，也是通常默认使用的模式便是 bridge 网络了。

在你安装，并启动 Docker Daemon 后，通过以下命令便可查看到默认的 bridge 网卡的信息：

复制

```
(MoeLove) ➜  ~ ifconfig docker0
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::xxx:be8  prefixlen 64  scopeid 0x20<link>
        ether 02:xx:0b:e8  txqueuelen 0  (Ethernet)
        RX packets 2175  bytes 347628 (339.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9639  bytes 11787827 (11.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

如果是使用默认的 bridge 网络，在启动容器后，所有的容器之间默认只能通过 IP 进行相互访问，除非你单独指定 `--link` 选项，或者是使用上篇介绍了 container 网络，共享网络堆栈后使用 localhost 访问已启动服务。

那么有没有更好的解决办法呢？

有，Docker 为我们提供了用户自定义 bridge 的方法。

用户自定义的 bridge 网络

Docker 提供了一组 network 对象，可供用户对网络资源进行管理，通过以下命令可以查看其支持的全部功能：

复制

```
(MoeLove) ➜  ~ docker network --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
```

使用默认配置创建 bridge 网络

使用 `docker network create 自定义网络名` 便可创建新的 bridge 网络了。例如：

复制

```
(MoeLove) ➜  ~ docker network create -d bridge moelove
fb24181d36d174dcbf1b84aea39544893ce01909ed2f7dfde079d01d2afe0638
```

创建 bridge 网络的时候，Docker 会自动创建一个对应的 bridge 网卡，可通过下面命令查看：

复制

```
(MoeLove) ➜  ~ ifconfig br-`docker network ls | grep moelove | awk '{print $1}'`
br-fb24181d36d1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.22.0.1  netmask 255.255.0.0  broadcast 172.22.255.255
        ether 02:42:c6:76:1f:88  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

注：在不同的机器上得到的结果会有所不同。

可以看到，当前创建了一个新的 bridge 网卡，分配的地址是 172.22.0.1/16，地址池等均是由 Docker 来控制分配的。

接下来启动一个容器，并使用此网络：

复制

```
(MoeLove) ➜  ~ docker run --name moe --rm -it --network moelove alpine sh
/ # hostname -i
172.22.0.2
/ # ip r
default via 172.22.0.1 dev eth0
172.22.0.0/16 dev eth0 scope link  src 172.22.0.2
```

启动的容器 IP 是从地址池中分配出来的，并且看路由信息会走刚才新生成的 bridge 网卡。

此时，在另一个终端窗口启动一个新的容器：

复制

```
(MoeLove) ➜  ~ docker run --name test --rm -it --network moelove alpine sh
/ # hostname -i
172.22.0.3
/ # ping -c 1 moe
PING moe (172.22.0.2): 56 data bytes
64 bytes from 172.22.0.2: seq=0 ttl=64 time=0.243 ms

--- moe ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.243/0.243/0.243 ms
```

可以看到新启动的容器内可以直接解析之前创建的容器的名称。**这是 Docker 内置 DNS 起的作用，在后续内容《Docker 内部 DNS 原理》中我们再详细聊。**

创建自定义 bridge 网络

在使用 Docker 创建 network 的时候，如果不加额外参数的话都使用 Docker 默认的配置。Docker 也提供了选项可供用户自己定义。

比如：我们可以自行指定地址池之类的，例如：

复制

```
(MoeLove) ➜  ~ docker network create -d bridge --gateway 192.168.130.1 --subnet
ba8357d7efc9826896987ed9b95f7fbe44d937d46ad5a2486994aa1e8833e3f8
(MoeLove) ➜  ~ ifconfig br-`docker network ls | grep moelove | awk '{print $1}'`
br-ba8357d7efc9: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.130.1  netmask 255.255.255.0  broadcast 192.168.130.255
        ether 02:42:f7:88:78:13  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

(MoeLove) ➜  ~ docker run --name test --rm -it --network moelove alpine sh
/ # hostname -i
192.168.130.2
/ # ip r
default via 192.168.130.1 dev eth0
192.168.130.0/24 dev eth0 scope link  src 192.168.130.2
```

通过 `--gateway` 和 `--subnet` 参数便可自行定义 bridge 网卡的配置了，当然还有一些其他的参数，你可以通过 `docker network create --help` 进行查看。

总结

本篇，我为你介绍了 Docker 自定义 bridge 网络相关的内容。

通过使用自定义 bridge 网络，可以启动 Docker 内置 DNS，提供更好的容器间的互联访问。

同时，Docker 也提供了很多参数，用于配置新创建的 bridge 网卡的信息。

下一篇，我将为你介绍如何灵活使用容器网络，让我们在使用 Docker 容器时候可以更加便捷。