Docker 内部 DN...

本篇是第七部分"网络篇"的第六篇。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，我为你介绍了 Docker 的一个重要组件 docker-proxy 的工作原理，本篇，我们来聊聊 Docker 内部 DNS 的原理。

在之前的内容《容器网络的灵活使用》中，我为你介绍过当使用自定义的 bridge 网络时，Docker 内置的 DNS 会帮助我们实现容器的互联，或者说可以帮我们实现容器名称的解析。

用一个例子来快速的复习一下：

复制

```
# 创建一个自定义的网络
(MoeLove) ➜  ~ docker network create course
1e0d66c191bf45ac1f74458828ec67c4d3143f604740277b66c1d19a098aeb78

# 启动一个容器，加入该自定义网络，并将容器命名为 redis
(MoeLove) ➜  ~ docker run --name redis --rm -d --network=course redis:alpine
fe8e6485dd053588ee7f501ad41be2c6ca5287f02b940690b9ac6b86314159ea

# 启动另一个容器，并加入该网络，尝试去 ping 之前容器的名字
(MoeLove) ➜  ~ docker run --rm -it --network=course  alpine
/ # ping -c 1 redis
PING redis (172.21.0.2): 56 data bytes
64 bytes from 172.21.0.2: seq=0 ttl=64 time=0.554 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.554/0.554/0.554 ms
```

可以看到，新启动的容器可以直接 ping 通之前启动容器的名字。这便要归功与 Docker 内置的 DNS 了。

工作原理

DNS 服务器地址

继续前面的例子，但为了能更加清晰，我们需要启动一个特权容器，来得到我们所需的信息：

复制

```
(MoeLove) ➜  ~ docker run --rm -it --network=course --privileged alpine
```

我们首先来查看容器中 /etc/resolv.conf 文件的内容，从这里可以得到我们在用的 DNS 服务器的配置：

```
/ # cat /etc/resolv.conf
nameserver 127.0.0.11
options ndots:0
```

可以看到，这里配置了 127.0.0.11，这是一个环回地址。既然已经确定了这是一个环回地址，那我们来看看 DNS Server 是如何为我们提供服务的。

服务端口

通过 netstat 命令来查看所有的连接。由于我们只是启动了一个 alpine 的镜像，实际并没有启动任何服务，但是这里可以看到有 127.0.0.11:35635 和 127.0.0.11:37358。地址恰好就是 127.0.0.11，所以我们推测这应该就是内置 DNS 监听的地址了。

```
/ # netstat  -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:35635        0.0.0.0:*               LISTEN
udp        0      0 127.0.0.11:37358        0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
```

尝试使用 dig 命令，手动验证域名解析的结果：

```
/ # apk add --no-cache -q bind-tools
/ # dig @127.0.0.11 -p 37358 +short redis
172.21.0.2
```

可以看到是可以使用它完成其他容器名称解析的。

实现原理

之前内容中，我已经为你介绍过 Docker 与 iptables 的关系。实际上在容器内，也是使用 iptables 来完成提供内置 DNS 服务相关功能的。

我们先来安装一个 iptables 工具，并查看容器内的 iptables 规则。

复制

```
/ # apk add --no-cache -q iptables
/ # iptables-save
# Generated by iptables-save v1.8.3 on Wed Mar 18 16:12:27 2020
*filter
:INPUT ACCEPT [7847:12644905]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [5659:326263]
COMMIT
# Completed on Wed Mar 18 16:12:27 2020
# Generated by iptables-save v1.8.3 on Wed Mar 18 16:12:27 2020
*nat
:PREROUTING ACCEPT [58:9908]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [35:2138]
:POSTROUTING ACCEPT [48:3018]
:DOCKER_OUTPUT - [0:0]
:DOCKER_POSTROUTING - [0:0]
-A OUTPUT -d 127.0.0.11/32 -j DOCKER_OUTPUT
-A POSTROUTING -d 127.0.0.11/32 -j DOCKER_POSTROUTING
-A DOCKER_OUTPUT -d 127.0.0.11/32 -p tcp -m tcp --dport 53 -j DNAT --to-destinatio
-A DOCKER_OUTPUT -d 127.0.0.11/32 -p udp -m udp --dport 53 -j DNAT --to-destinatio
-A DOCKER_POSTROUTING -s 127.0.0.11/32 -p tcp -m tcp --sport 35635 -j SNAT --to-so
-A DOCKER_POSTROUTING -s 127.0.0.11/32 -p udp -m udp --sport 37358 -j SNAT --to-so
COMMIT
# Completed on Wed Mar 18 16:12:27 2020
```

可以看到，当出口目标为 127.0.0.11 也就是 /etc/resolv.conf 中配置的 DNS 服务器地址时，会使用 iptables 为请求提供 SNAT 和 DNAT 的支持，将请求转发至具体的内置 DNS 服务器上，以完成此功能。

而这些，对用户而言都是透明的。用户正常情况下是无需关注此逻辑的。当然，当你使用 docker-compose 之类的工具进行容器编排时，也不经意的都使用了内置 DNS 提供的服务。

总结

本篇，我为你介绍了 Docker 内置 DNS 的工作原理。Docker 通过在容器内添加的 iptables 规则，实现了自动将 DNS 解析请求转发到内置 DNS 的功能。

下一篇，我们来总结下“网络篇”中我们所学到的内容，以及对 Docker 网络进一步加深认识。