> 本文由 简悦 SimpRead 转码， 原文地址 www.imooc.com

前面两篇文章介绍了 Docker 网络情况，这篇文章介绍一下 Link 的原理。这里说的 Link 是在 Docker 容器创建的过程中通过 `--link` 参数将新创建出来的 Docker 容器和已有的容器之间串讲一个安全通道用来做数据交互。

Link 的使用场景还是很常见的，比如我们线上应用有一个 web 应用以 Docker 容器运行，有一个数据库（MySQL）也以 Docker 容器运行，由于 web 应用需要访问数据库的数据，那么我们就可以在这两个容器之间使用 Link 连接起来。

1. Link 使用

Link 的使用比较简单，我们这里演示一下。首先运行一个 MySQL 的 Docker 容器。

```
[root@docker1 ~]
e47e603ffb17f4b42d8841ff26d3b93935eed4cb4e3155ae901c0a3afce37b45
[root@docker1 ~]
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                               NAMES
e47e603ffb17        mysql:latest        "docker-entrypoint.s…"   6 seconds ago
    Up 5 seconds        33060/tcp, 0.0.0.0:3307->3306/tcp   mysql
```

然后我们创建一个 busybox 的 Docker 容器，并通过 telnet 连接 MySQL 的 Docker 容器。

```
[root@docker1 ~]
/
telnet: can't connect to remote host (172.17.0.2): Connection refused
/
Connected to mysql
J
5.1.jJXq/%
   p@R|Iccaching_sha2_password
```

其中 busybox 容器的启动参数里面的 `--link mysql:mysql` 就是将我们新建出来的 busybox 容器和名字叫 mysql 的 Docker 容器建立一个 link 通道。`--link` 的参数格式为 `--link <name or id>:alias`，第一个参数是目标容器的名字或者 ID，第二个 alias 相当于我们在 busybox Docker 容器中访问 MySQL Docker 容器的 host。

为了表示我们确实是通过 link 连通了这两个 Docker 容器，我们不带 link 参数创建一个 busybox 容器出来，并尝试访问 MySQL 的 Docker 容器，然后直接提示无法对 mysql 做 dns 解析。

```
[root@docker1 ~]
/
telnet: bad address 'mysql'
```

那么 link 究竟做了什么呢?

2. hosts 文件修改

通过上面的例子我们发现没有加 link 参数时，提示无法做 dns 解析，那么直觉告诉我们这个操作可能和 /etc/hosts 文件相关，我们比较一下加了 link 参数和没有加 link 参数的两个容器的 /etc/hosts 文件。

```
/
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  mysql e47e603ffb17
172.17.0.3  d73dc6529032
```

```
/
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.4  f1090a6f3bf2
```

我们可以看到加了 link 参数的容器的 hosts 文件中多了一条记录 `172.17.0.2 mysql e47e603ffb17`，这条记录正是告诉我们如何访问 mysql。

3. 环境变量

当两个容器通过 `--link` 建立连接后，会在接收容器中额外设置一些环境变量以保存源容器的一些信息。下面是我们 busybox 中保存的关于源容器 MySQL 的一些环境变量。主要包括一些 IP、端口和网络协议信息。

```
/
MYSQL_PORT_33060_TCP=tcp:
MYSQL_ENV_MYSQL_MAJOR=8.0
MYSQL_PORT_3306_TCP_ADDR=172.17.0.2
MYSQL_ENV_MYSQL_ROOT_PASSWORD=123456
MYSQL_ENV_GOSU_VERSION=1.7
MYSQL_PORT_3306_TCP_PORT=3306
MYSQL_PORT_3306_TCP_PROTO=tcp
MYSQL_PORT_33060_TCP_ADDR=172.17.0.2
MYSQL_PORT=tcp:
MYSQL_PORT_3306_TCP=tcp:
MYSQL_PORT_33060_TCP_PORT=33060
MYSQL_ENV_MYSQL_VERSION=8.0.19-1debian9
MYSQL_PORT_33060_TCP_PROTO=tcp
MYSQL_NAME=/busybox/mysql
```

4. iptables

在接收容器上设置了环境变化和更改了 /etc/hosts 文件之后，接收容器仅仅是得到了源容器的相关信息，比如 IP、端口等，但是并不能表示两个容器之间可以互相通信。那么网络通信如何来保证呢?

如果对前面的那篇 《Docker 网络初探》还有印象的话，这个时候应该可以想到 iptables，事实上 Docker 也确实是这么做的。通过 iptables-save 我们可以观察到下面两天 ***filter*** 规则。

```
-A DOCKER -s 172.17.0.2/32 172.17.0.3/32 -i docker0 -o docker0 -p tcp -m tcp --
dport 3306 -j ACCEPT
-A DOCKER -s 172.17.0.3/32 172.17.0.2/32 -i docker0 -o docker0 -p tcp -m tcp --
dport 3306 -j ACCEPT
```

这两条规则确保了我们的 busybox 容器在源容器（MySQL 容器）的 tcp/3306 端口上通信的流量不会被丢掉，从而保证了接收容器可以顺利地从源容器中获取到想要的数据。

5. 总结

本节简单介绍了 Docker 的 link 技术工作原理，可以看出来原理还是比较简单的。

值得一提的是，现在官方已经不建议使用 `--link` 来进行网络通信了。

> **Warning**: The `--link` flag is a legacy feature of Docker. It may eventually be removed. Unless you absolutely need to continue using it, we recommend that you use user-defined networks to facilitate communication between two containers instead of using `--link`. One feature that user-defined networks do not support that you can do with `--link` is sharing environment variables between containers. However, you can use other mechanisms such as volumes to share environment variables between containers in a more controlled way.

简单翻译一下:

> 警告：`--link` 参数是 Docker 早期的遗留特性，可能最终会被移除掉。除非你一定要使用它，否则我们建议你使用自定义网络的方式来实现多个 container 之间的网络通信。自定义网络相比 `--link` 的一个弊端是无法共享环境变量，但是你可以通过类似在多个容器中挂载同一个 volume 的方式来实现这个需求。

事实上，当我们接触到 Kubernetes 之后，我们就会知道 link 这种方式是多么的简陋。

}