Docker 与 iptab...

本篇是第七部分“网络篇”的第三篇。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，我为你介绍了如何灵活的使用容器网络。本篇，我们来学习 Docker 与 iptables 之间的联系。

Docker 能为我们提供如此强大和灵活的网络能力，很大程度上要归功与 iptables 的结合。在使用时，你可能没有太关注到 iptables 的作用，这是因为 Docker 已经帮我们自动完成了。

复制

```
(MoeLove) ➜  ~ dockerd --help |grep iptables
      --iptables                          Enable addition of iptables rules
```

Docker Daemon 有个 `--iptables` 的参数，便是用来控制是否要启用 iptables 规则的，默认已经设置成了开启（true）。所以通常我们不会过于关注到它的工作。

本篇，为了避免环境的干扰，我将使用 Docker in Docker 的环境来为你进行介绍，通过如下方式启动该环境：

复制

```
(MoeLove) ➜  ~ docker run --rm -d --privileged docker:dind
ceb59125626e4ced03f3e4fa73a09b50633ce4f001938f7a8e04804d99420efa
```

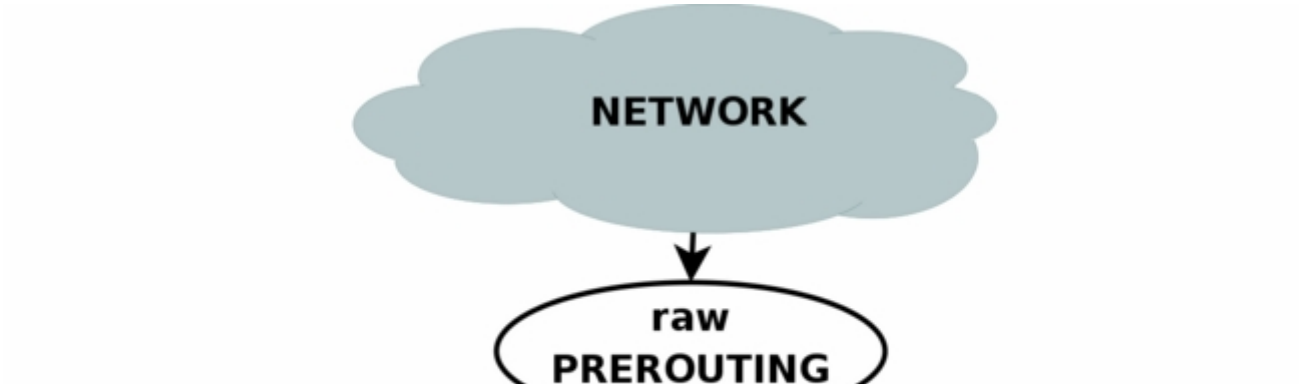
iptables 基础

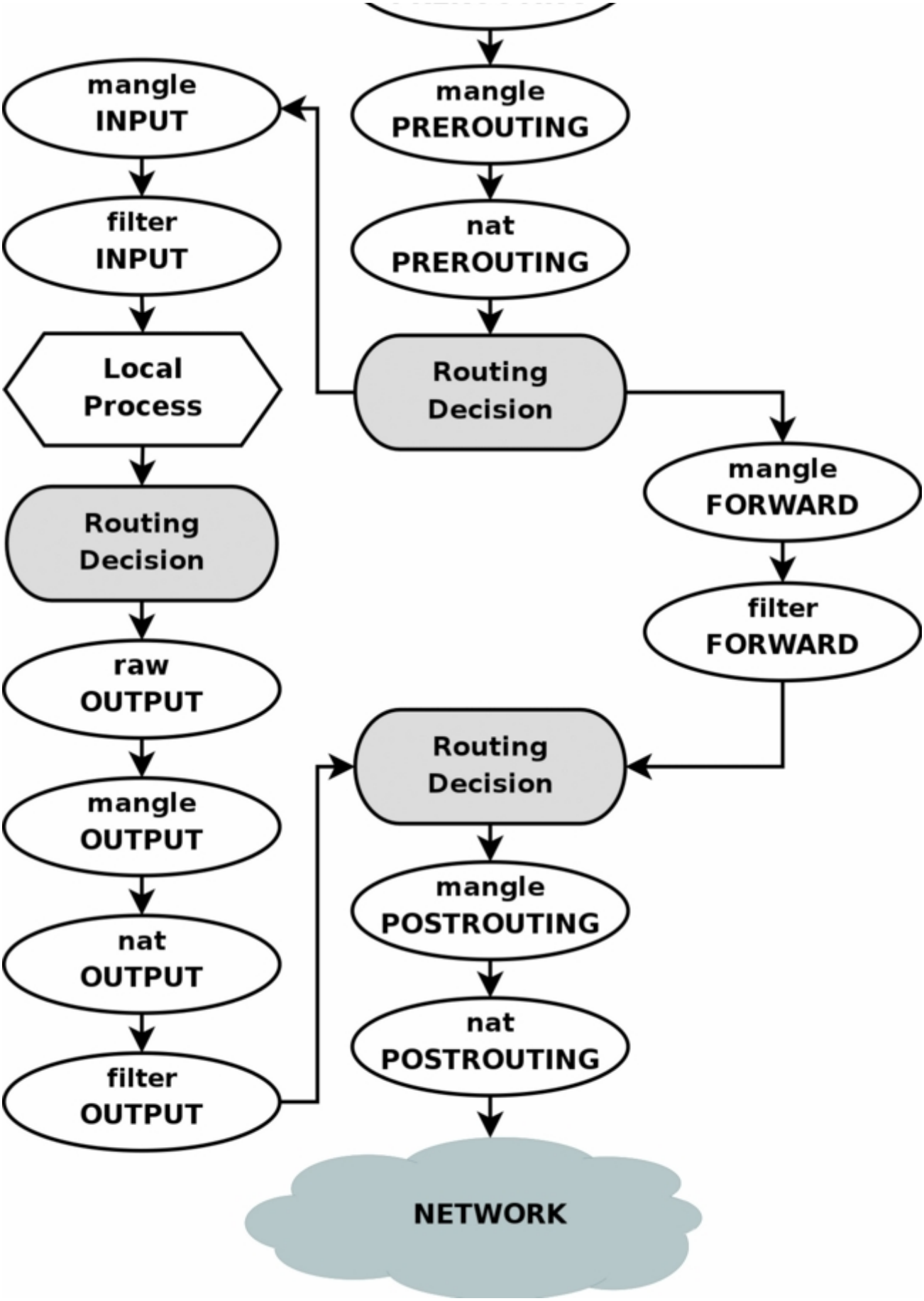
iptables 是一个用于配置 Linux 内核防火墙的工具，可用于检测、修改转发、重定向以及丢弃 IPv4 数据包。它使用了内核的 ip_tables 的功能，所以需要 Linux 2.4+ 版本的内核。

同时，iptables 为了便于管理，所以按照不同的目的组织了多张**表**；每张表中又包含了很多预定义的**链**；每个链中包含着顺序遍历的**规则**；这些规则中又定义了动作的匹配规则和**目标**。

对于用户而言，我们通常需要交互的就是**链**和**规则**了。

理解 iptables 的主要工作流程有一张比较经典的图：





图片来源：www.frozentux.net

上面的小写字母是**表**，大写字母则表示**链**，从任何网络端口 进来的每一个 IP 数据包都要从上到下的穿过这张图。

> 引用自 ArchWiki

Docker 网络与 iptables

前面已经对 iptables 有了个大体介绍，对 iptables 更深入的学习，建议在终端下执行 `man iptables` 查看相关文档。

接下来我们直接看看 Docker 网络与 iptables 具体有什么样的联系。

关闭 Docker 的 iptables 支持

在本文开头已经为你介绍过 Docker Daemon 存在一个 `--iptables` 的参数，用于控制是否使用 iptables。我们使用以下命令启动一个 Docker Daemon 并关闭 iptables 支持。

复制

```
(MoeLove) ➜  ~ docker run --rm -d --privileged docker:dind  dockerd --iptables=fa
cb3396e9a8b61aa024b5c9457fcd2b26cf85f0d516bc92ff9934497ac3b23e9d
```

进入此容器，并查看其所有 iptables 规则：

复制

```
(MoeLove) ➜  ~ docker exec -it $(docker ps -ql) sh
/ # iptables-save
# Generated by iptables-save v1.8.3 on Mon Feb 24 16:44:12 2020
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [1:40]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A DOCKER-USER -j RETURN
COMMIT
# Completed on Mon Feb 24 16:44:12 2020
# Generated by iptables-save v1.8.3 on Mon Feb 24 16:44:12 2020
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
COMMIT
# Completed on Mon Feb 24 16:44:12 2020
```

可以看到，当 Docker Daemon 加了 `--iptables=false` 的参数时，增加了一个 DOCKER-USER 的链，以及多了两条有关的规则。

你可能会好奇，为什么在已经加了 `--iptables=false` 的情况下，还是会有对应的修改。这其实是 Docker 预留一个链。

具体的实现均在 docker/libnetwork 下，以下是关于 DOCKER-USER 链的相关代码：

复制

```
const userChain = "DOCKER-USER"

func arrangeUserFilterRule() {
    _, err := iptables.NewChain(userChain, iptables.Filter, false)
    if err != nil {
        logrus.Warnf("Failed to create %s chain: %v", userChain, err)
        return
    }

    if err = iptables.AddReturnRule(userChain); err != nil {
        logrus.Warnf("Failed to add the RETURN rule for %s: %v", userChain, err)
        return
    }

    err = iptables.EnsureJumpRule("FORWARD", userChain)
    if err != nil {
        logrus.Warnf("Failed to ensure the jump rule for %s: %v", userChain, err)
    }
}
```

可以看到链名称是固定在代码中的，同时会先后新建此链，以及增加相关规则。

最后，需要注意的是：DOCKER-USER 链允许用户以持久化的方式自行配置网络规则，而不必担心 Docker 的重启或者关闭。Docker 不会删除或修改任何 DOCKER-USER 链上的规则。

开启 Docker 的 iptables 支持

使用以下命令启动一个 Docker Daemon，这里没有显示的传递 `--iptables` 选项，因为默认就是 true。

复制

```
(MoeLove) ➜  ~ docker run --rm -d --privileged docker:dind
c464c5c08ecdf9129afbf217c6462236089fe0a1d11dfe7700c2985a04d8d216
```

查看其 iptables 规则：

复制

```
(MoeLove) ➜  ~ docker exec -it $(docker ps -ql) sh
/ # iptables-save
# Generated by iptables-save v1.8.3 on Mon Feb 24 17:06:48 2020
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
# Completed on Mon Feb 24 17:06:48 2020
# Generated by iptables-save v1.8.3 on Mon Feb 24 17:06:48 2020
*nat
:PREROUTING ACCEPT [18:2796]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [1:40]
:POSTROUTING ACCEPT [1:40]
:DOCKER - [0:0]
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -s 172.18.0.0/16 ! -o docker0 -j MASQUERADE
-A DOCKER -i docker0 -j RETURN
COMMIT
# Completed on Mon Feb 24 17:06:48 2020
```

可以看到，它比刚才关闭 iptables 支持时多了几条链和对应的规则。尤其是增加了一些转发规则。

同时你可以看到其中多了一些与 docker0 相关的规则，还记得前两篇中我为你介绍 bridge 网络时介绍过的 docker0 吗？

事实上它能够这么灵活是离不开 iptables 的支持的。

现在，我们运行一个容器，并进行端口映射，试试看启动容器时，Docker 对 iptables 做了什么。

```
(MoeLove) ➜  ~ docker exec -it $(docker ps -ql) sh
/ # docker run -P --rm -d redis:alpine
Unable to find image 'redis:alpine' locally
alpine: Pulling from library/redis
c9b1b535fdd9: Pull complete
8dd5e7a0ba4a: Pull complete
e20c1cdf5aef: Pull complete
f06a0c1e566e: Pull complete
230b5c8df708: Pull complete
0cb9ac88f5bf: Pull complete
Digest: sha256:cb9783b1c39bb34f8d6572406139ab325c4fac0b28aaa25d5350495637bb2f76
Status: Downloaded newer image for redis:alpine
a97821d655552948df33eecd948df9784b6d05e25420b45f13c3a368c3d6f956
/ # docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
a97821d65555        redis:alpine        "docker-entrypoint.s…"   7 seconds ago
```

之后再次执行 iptables-save，对比当前的结果与上次的差别：

```
 *filter
+-A DOCKER -d 172.18.0.2/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 6379 -j
 *nat
+-A POSTROUTING -s 172.18.0.2/32 -d 172.18.0.2/32 -p tcp -m tcp --dport 6379 -j MA
+-A DOCKER ! -i docker0 -p tcp -m tcp --dport 32768 -j DNAT --to-destination 172.1
```

Docker 分别在 filter 表和 nat 表增加了规则。它的具体含义如下：

filter 表中新增的这条规则表示：在自定义的 DOCKER 链中，对于目标地址是 172.18.0.2 且不是从 docker0 进入的且从 docker0 出去的，目标端口是 6379 的 TCP 协议则接收。

简单点来说就是，放行通过 docker0 流出的，目标为 172.18.0.2:6379 的 TCP 协议的流量。

nat 表中这两条规则的表示：

- 为 172.18.0.2 上目标端口为 6379 的流量执行 MASQUERADE 动作（这里就简单地将它理解为 SNAT 也可以）；
- 在自定义的 DOCKER 链中，如果入口不是 docker0 并且目标端口是 32768 则进行 DNAT 动作，将目标地址转换为 172.18.0.2:6379。简单点来说，这条规则就是为我们提供了 Docker 容器端口转发的能力，将访问主机本地 32768 端口流量的目标地址转换为 172.18.0.2:6379。

当然，要提供完整的访问能力，也需要和其他前面列出的其他规则共同配合才能完成，我们在下篇来具体进行介绍。

总结

本篇，我为你介绍了 dockerd 的 `--iptables` 参数的作用，以及对比了是否使用此选项的差异。通过查看 Docker 的源码可以看到 Docker Daemon 启动时，默认在 iptables 中增加了一些特定的规则和自定义链，以此来实现 Docker 容器网络相关的功能。

下篇，我们来具体聊聊如何手动进行容器网络的管理，通过该篇内容来更深入的了解 Docker 是如何使用 iptables 为容器网络提供各类特性的。