太长不想读

请直接看解决 UFW 和 Docker 的问题。

问题

UFW 是 Ubuntu 上很流行的一个 iptables 前端，可以非常方便的管理防火墙的规则。但是当安装了 Docker，UFW 无法管理 Docker 发布出来的端口了。

具体现象是：

1. 在一个对外提供服务的服务器上启用了 UFW，并且默认阻止所有未被允许的传入连接。
2. 运行了一个 Docker 容器，并且使用 `-p` 选项来把该容器的某个端口发布到服务器的所有 IP 地址上。比如：`docker run -d --name httpd -p 0.0.0.0:8080:80 httpd:alpine` 将会运行一个 httpd 服务，并且将容器的 `80` 端口发布到服务器的 `8080` 端口上。
3. UFW 将不会阻止所有对 `8080` 端口访问的请求，用命令 `ufw deny 8080` 也无法阻止外部访问这个端口。

这个问题其实挺严重的，这意味着本来只是为了在内部提供服务的一个端口被暴露在公共网络上。

在网络上搜索 'ufw docker' 可以发现很多的讨论：

- https://github.com/moby/moby/issues/4737
- https://forums.docker.com/t/running-multiple-docker-containers-with-ufw-and-iptables-false/8953
- https://www.techrepublic.com/article/how-to-fix-the-docker-and-ufw-security-flaw/
- https://blog.viktorpetersson.com/2014/11/03/the-dangers-of-ufw-docker.html
- https://askubuntu.com/questions/652556/uncomplicated-firewall-ufw-is-not-blocking-anything-when-using-docker
- https://chjdev.com/2016/06/08/docker-ufw/
- https://askubuntu.com/questions/652556/uncomplicated-firewall-ufw-is-not-blocking-anything-when-using-docker
- https://my.oschina.net/abcfy2/blog/539485
- https://www.v2ex.com/amp/t/466666
- https://blog.36web.rocks/2016/07/08/docker-behind-ufw.html
- ...

基本上可以找到的解决办法就是首先禁用 docker 的 iptables 功能，但这也意味着放弃了 docker 的网络管理功能，很典型的现象就是容器将无法访问外部网络。在有的文章中也提到了可以在 UFW 的配置文件中手工添加一条规则，比如 `-A POSTROUTING ! -o docker0 -s 172.17.0.0/16 -j MASQUERADE`。但这也只是允许了 `172.17.0.0/16` 这个网络。如果有了新增的网络，我们也必须手工再为新增的网络添加这样类似的 iptables 规则。

期望的目标

目前网络上的解决方案都非常类似，而且也不优雅，我希望一个新的解决方案可以：

1. 不要禁用 Docker 的 iptables，像往常一样由 Docker 来管理自己的网络。这样有任何新增的 Docker 网络时都无需手工维护 iptables 规则，也避免了在 Docker 中禁用 iptables 之后可能带来的副作用。
2. 公共网络不可以访问 Docker 发布出来的端口，即使是使用类似 `-p 0.0.0.0:8080:80` 的选项把端口发布在所有的 IP 地址上。容器之间、内部网络之间都可以正常互相访问，只有公共网络不可以访问。 虽然可以让 Docker 把容器的某一个端口映射到服务器的私有 IP 地址上，这样公共网络上将不会访问到这个端口。但是这个服务器可能有多个私有 IP 地址，这些私有 IP 地址可能也会发生变化。
3. 可以很方便的允许公共网络直接访问某个容器的端口，而无需额外的软件和配置。就像是用 `ufw allow 8080` 这样允许外部访问 8080 端口，然后用 `ufw delete allow 8080` 就不再允许外部访问。

如何做？

撤销原先的修改

如果已经按照目前网络上搜索到解决方案修改过了，请先修改回来，包括：

1. 启用 Docker 的 iptables 功能，删除所有类似 `--iptables=false` 的修改，包括 `/etc/docker/daemon.json` 配置文件。
2. UFW 的默认 `FORWARD` 规则改回默认的 `DROP`，而非 `ACCEPT`。
3. 删除 UFW 配置文件 `/etc/ufw/after.rules` 中与 Docker 网络相关的规则。
4. 如果修改了 Docker 相关的配置文件，重启 Docker。稍后还要修改 UFW 的配置，可以一并重启。

解决 UFW 和 Docker 的问题

目前新的解决方案只需要修改一个 UFW 配置文件即可，Docker 的所有配置和选项都保持默认。

修改 UFW 的配置文件 `/etc/ufw/after.rules`，在最后添加上如下规则：

```
# BEGIN UFW AND DOCKER
*filter
:ufw-user-forward - [0:0]
:ufw-docker-logging-deny - [0:0]
:DOCKER-USER - [0:0]
-A DOCKER-USER -j ufw-user-forward

-A DOCKER-USER -j RETURN -s 10.0.0.0/8
-A DOCKER-USER -j RETURN -s 172.16.0.0/12
-A DOCKER-USER -j RETURN -s 192.168.0.0/16

-A DOCKER-USER -p udp -m udp --sport 53 --dport 1024:65535 -j RETURN

-A DOCKER-USER -j ufw-docker-logging-deny -p tcp -m tcp --tcp-flags
FIN,SYN,RST,ACK SYN -d 192.168.0.0/16
-A DOCKER-USER -j ufw-docker-logging-deny -p tcp -m tcp --tcp-flags
FIN,SYN,RST,ACK SYN -d 10.0.0.0/8
-A DOCKER-USER -j ufw-docker-logging-deny -p tcp -m tcp --tcp-flags
FIN,SYN,RST,ACK SYN -d 172.16.0.0/12
-A DOCKER-USER -j ufw-docker-logging-deny -p udp -m udp --dport 0:32767 -
d 192.168.0.0/16
-A DOCKER-USER -j ufw-docker-logging-deny -p udp -m udp --dport 0:32767 -
d 10.0.0.0/8
-A DOCKER-USER -j ufw-docker-logging-deny -p udp -m udp --dport 0:32767 -
d 172.16.0.0/12

-A DOCKER-USER -j RETURN

-A ufw-docker-logging-deny -m limit --limit 3/min --limit-burst 10 -j
LOG --log-prefix '[UFW DOCKER BLOCK] '
-A ufw-docker-logging-deny -j DROP

COMMIT
# END UFW AND DOCKER
```

然后重启 UFW，`sudo systemctl restart ufw`。现在外部就已经无法访问 Docker 发布出来的任何端口了，但是容器内部以及私有网络地址上可以正常互相访问，而且容器也可以正常访问外部的网络。**可能由于某些未知原因，重启 UFW 之后规则也无法生效，请重启服务器。**

如果希望允许外部网络访问 Docker 容器提供的服务，比如有一个容器的服务端口是 `80`。那就可以用以下命令来允许外部网络访问这个服务：

```
ufw route allow proto tcp from any to any port 80
```

这个命令会允许外部网络访问所有用 Docker 发布出来的并且内部服务端口为 `80` 的所有服务。

请注意，这个端口 `80` 是容器的端口，而非使用 `-p 0.0.0.0:8080:80` 选项发布在服务器上的 `8080` 端口。

如果有多个容器的服务端口为 80，但只希望外部网络访问某个特定的容器。比如该容器的私有地址为 `172.17.0.2`，就用类似下面的命令：

```
ufw route allow proto tcp from any to 172.17.0.2 port 80
```

如果一个容器的服务是 UDP 协议，假如是 DNS 服务，可以用下面的命令来允许外部网络访问所有发布出来的 DNS 服务：

```
ufw route allow proto udp from any to any port 53
```

同样的，如果只针对一个特定的容器，比如 IP 地址为 `172.17.0.2`：

```
ufw route allow proto udp from any to 172.17.0.2 port 53
```

解释

在新增的这段规则中，下面这段规则是为了让私有网络地址可以互相访问。通常情况下，私有网络是比公共网络更信任的。

```
-A DOCKER-USER -j RETURN -s 10.0.0.0/8
-A DOCKER-USER -j RETURN -s 172.16.0.0/12
-A DOCKER-USER -j RETURN -s 192.168.0.0/16
```

下面的规则是为了可以用 UFW 来管理外部网络是否允许访问 Docker 容器提供的服务，这样我们就可以在一个地方来管理防火墙的规则了。

```
-A DOCKER-USER -j ufw-user-forward
```

例如，我们要阻止一个 IP 地址为 172.17.0.9 的容器内的所有对外连接，也就是阻止该容器访问外部网络，使用下列命令

```
ufw route deny from 172.17.0.9 to any
```

下面的规则阻止了所有外部网络发起的连接请求，但是允许内部网络访问外部网络。对于 TCP 协议，是阻止了从外部网络主动建立 TCP 连接。对于 UDP，是阻止了所有小余端口 `32767` 的访问。为什么是这个端口的？由于 UDP 协议是无状态的，无法像 TCP 那样阻止发起建立连接请求的握手信号。在 GNU/Linux 上查看文件 `/proc/sys/net/ipv4/ip_local_port_range` 可以看到发出 TCP/UDP 数据后，本地源端口的范围，默认为 `32768 60999`。当从一个运行的容器对外访问一个 UDP 协议的服务时，本地端口将会从这个端口范围里面随机选择一个，服务器将会把数据返回到这个随机端口上。所以，我们可以假定所有容器内部的 UDP 协议的监听端口都小余 `32768`，不允许外部网络主动连接小余 `32768` 的 UDP 端口。

```
-A DOCKER-USER -j DROP -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -d
192.168.0.0/16
-A DOCKER-USER -j DROP -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -d
10.0.0.0/8
-A DOCKER-USER -j DROP -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -d
172.16.0.0/12
-A DOCKER-USER -j DROP -p udp -m udp --dport 0:32767 -d 192.168.0.0/16
-A DOCKER-USER -j DROP -p udp -m udp --dport 0:32767 -d 10.0.0.0/8
-A DOCKER-USER -j DROP -p udp -m udp --dport 0:32767 -d 172.16.0.0/12

-A DOCKER-USER -j RETURN
```

如果一个容器在接受数据的时候，端口号没有遵循操作系统的设定，也就是说最小端口号要小余 `32768`。比如运行了一个 Dnsmasq 的容器，Dnsmasq 用于接受数据的最小端口号默认是 `1024`。那可以用下面的命令来允许 Dnsmasq 这个容器使用一个更大的端口范围来接受数据。

```
ufw route allow proto udp from any port 53 to any port 1024:65535
```

因为 DNS 是一个非常常见的服务，所以已经有一条规则用于允许使用一个更大的端口范围来接受 DNS 数据包

选择 `ufw-user-forward` 而不是 `ufw-user-input` 的原因

使用 `ufw-user-input`

优点：

使用的 UFW 命令比较简单，也比较容易理解，而且也支持老版本的 Ubuntu

比如，允许公众网络访问一个已经发布出来的容器端口 `8080`，使用命令：

```
ufw allow 8080
```

缺点：

不仅仅是暴露了已经发布的容器端口，也暴露了主机上的端口。

比如，如果在主机上运行了一个端口为 `8080` 的服务。命令 `ufw allow 8080` 允许了公共网络访问这个服务，也允许了访问所有已经发布的容器端口为 `8080` 的服务。但是我们可能只是希望保留主机上的这个服务，或者是运行在容器里面的服务，而不是两个同时暴露。

为了避免这个问题，我们可能需要使用类似下面的命令来管理已经发布的容器端口：

```
ufw allow proto tcp from any to 172.16.0.3 port 8080
```

使用 `ufw-user-forward`

优点：

不会因为同一条命令而同时暴露主机和容器里面的服务。

比如，如果我们希望暴露所有容器端口为 `8080` 的服务，使用下面的命令：

```
ufw route allow 8080
```

现在公共网络可以访问所有容器端口为 `8080` 的已经发布的服务，但是运行在主机上的 `8080` 服务仍然不会被公开。如果我们希望公开主机上的 `8080` 端口，可以执行下面的命令：

```
ufw allow 8080
```

缺点：

不支持老版本的 Ubuntu，而且命令的使用上可能也会比较复杂。

结论

如果我们正在使用老版本的 Ubuntu，我们可以使用 `ufw-user-input`。但是要小心避免把不该暴露的服务暴露出去。

如果正在使用支持 `ufw route` 命令的新版本的 Ubuntu，我们最好使用 `ufw-user-forward`，并且使用 `ufw route` 来管理与容器相关的防火墙规则。

`ufw-docker` 工具

现在这个脚本也支持 Docker Swarm。

安装

下载 `ufw-docker` 脚本

```
sudo wget -O /usr/local/bin/ufw-docker \
  https://github.com/chaifeng/ufw-docker/raw/master/ufw-docker
chmod +x /usr/local/bin/ufw-docker
```

使用下列命令来修改 ufw 的 `after.rules` 文件

```
ufw-docker install
```

这个命令做了以下事情:

- 备份文件 `/etc/ufw/after.rules`
- 把 UFW 和 Docker 的相关规则添加到文件 `after.rules` 的末尾

为 Docker Swarm 环境安装

仅仅可以在管理节点上使用 `ufw-docker` 这个脚本来管理防火墙规则。

- 在所有的节点上修改 `after.rules` 这个文件，包括管理节点和工作节点
- 在管理节点上部署这个脚本

运行在 Docker Swarm 模式下，这个脚本将会创建一个全局服务 `ufw-docker-agent` 。这个镜像 chaifeng/ufw-docker-agent 是由本项目自动构建的。

使用方法

显示帮助

```
ufw-docker help
```

检查 UFW 配置文件中防火墙规则的安装

```
ufw-docker check
```

更新 UFW 的配置文件，添加必要的防火墙规则

```
ufw-docker install
```

显示当前防火墙允许的转发规则

```
ufw-docker status
```

列出所有和容器 `httpd` 相关的防火墙规则

```
ufw-docker list httpd
```

暴露容器 `httpd` 的 `80` 端口

```
ufw-docker allow httpd 80
```

暴露容器 `httpd` 的 `443` 端口，且协议为 `tcp`

```
ufw-docker allow httpd 443/tcp
```

把容器 `httpd` 的所有映射端口都暴露出来

```
ufw-docker allow httpd
```

删除所有和容器 `httpd` 相关的防火墙规则

```
ufw-docker delete allow httpd
```

删除容器 `httpd` 的 `tcp` 端口 `443` 的规则

```
ufw-docker delete allow httpd 443/tcp
```

暴露服务 `web` 的 `80` 端口

```
docker service create --name web --publish 8080:80 httpd:alpine

ufw-docker service allow web 80
# 或者
ufw-docker service allow web 80/tcp
```

删除与服务 `web` 相关的规则

```
ufw-docker service delete allow web
```

试试

我们使用 Vagrant 来创建一个本地的测试环境。

运行下面的命令来创建 1 个 master 节点和 2 个 workder 节点

```
vagrant up
```

登录到 master 节点

```
vagrant ssh master
```

登录后，创建 `web` 服务

```
docker service create --name web --publish 8080:80 httpd:alpine
```

我们应该无法从我们的主机上访问这个 `web` 服务

```
curl -v http://192.168.56.131:8080
```

在 master 节点上，运行下面的命令来允许公共访问 `web` 服务端 `80` 端口。

```
sudo ufw-docker service allow web 80
```

现在我们可以在我们的主机上访问这个 `web` 服务了

```
curl 'http://192.168.56.13{0,1,2}:8080'
```

讨论

- What is the best practice of docker + ufw under Ubuntu - Stack Overflow
- docker and ufw serious problems · Issue #4737 · moby/moby