Docker and IPtables ()

TL;DR; By default, `docker` daemon appends `iptables` rules for forwarding. For this, it uses a filter chain named `DOCKER`.

```
Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0
...

Chain DOCKER (1 references)
target     prot opt source               destination
```

Moreover, when you tell docker to expose a port of a container, it exposes it to the entire world, breaking your possibly existing `iptables` rules.

So.. if you are running `docker` on a host that already have an iptables based firewall, you should probably set `--iptables=false`.

What are you talking about?

Let's take an example. You want to start nginx and bind `containerPort` `80` to `hostPort` `9090`:

```
docker run --name some-nginx -d -p 9090:80 nginx
```

What it does behind the scene is adding an `iptables` rule to the `DOCKER` filter chain:

```
Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0
...

Chain DOCKER (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            172.17.0.2           tcp dpt:9090 <-- this was added when run
ning the container
```

Now port `9090` is available from the entire world. Why? Because we're listening `9090` on any IP addresses (`*`) and because of the forwarding rules that are dynamically added in the `DOCKER` filter chain. Note that docker's forward rules permit all external source IPs by default.

You probably don't want that.

Exposing ports locally

You might want to publish ports just locally and not to `*`, for internal use. Let's read the documentation of `docker run`:

```
-p=[]      : Publish a container's port or a range of ports to the host
               format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | contai
nerPort
               Both hostPort and containerPort can be specified as a range of ports.
               When specifying ranges for both, the number of container ports in the range must match
 the number of host ports in the range. (e.g., `-p 1234-1236:1234-1236/tcp`)
               (use 'docker port' to see the actual mapping)
```

As you can see, you can bind the `hostPort` to an IP.

```
docker run --name some-nginx -d -p 127.0.0.1:9090:80 nginx
```

```
# BEFORE
netstat -an | grep 9090
tcp6       0      0 :::9090                 :::*                    LISTEN
# AFTER
netstat -an | grep 9090
tcp        0      0 127.0.0.1:9090          0.0.0.0:*               LISTEN
```

Better.

Docker, stop messing with my iptables rules!

Let's say you are using `docker` on a server available on the Internet. You already have an `iptables` based firewall configured. Personally, I'm using uif (https://github.com/cajus/uif) which is a very powerful perl script available (https://packages.debian.org/jessie/uif) in `debian`. Have a look at a config example (https://github.com/cajus/uif/blob/master/docs/uif.conf.IPv4%2B6.tmpl).

To tell `docker` to never make changes to your system `iptables` rules, you have to set `--iptables=false` when the daemon starts.

For `sysvinit` and `upstart` based systems, you can edit `/etc/default/docker`. For systemd, you can do that:

```
mkdir /etc/systemd/system/docker.service.d
cat << EOF > /etc/systemd/system/docker.service.d/noiptables.conf
[Service]
ExecStart=
ExecStart=/usr/bin/docker daemon -H fd:// --iptables=false
EOF
systemctl daemon-reload
```

Now reload your firewall and restart `docker` daemon. You can see that the chain named `DOCKER` and the references to it in chain `FORWARD` (policy `DROP`) disappeared.

Configure iptables to work with docker

If you're still using the Ethernet bridge (https://docs.docker.com/articles/networking/#bridge-building) created by `docker` and named `docker0`, you can set the following rules for forwarding:

```
# just an example. It implies that your host Ethernet NIC is eth0
-A FORWARD -i docker0 -o eth0 -j ACCEPT
-A FORWARD -i eth0 -o docker0 -j ACCEPT
```

Now if you want to expose TCP port `10000` of a running container to the world, this container must expose port to any IP (`*`) on host side:

```
docker run --name some-nginx -d -p 10000:80 nginx

netstat -an | grep 10000
tcp6       0      0 :::10000                :::*                    LISTEN
```

Then you can add this firewall rule to allow the world to access your container through the forwarding rules:

```
-A INPUT -p tcp -m tcp --dport 10000 -s 0.0.0.0/0 -j ACCEPT
```

References

- https://docs.docker.com/reference/run/ (https://docs.docker.com/reference/run/)
- https://docs.docker.com/articles/systemd/ (https://docs.docker.com/articles/systemd/)
- https://docs.docker.com/articles/networking/ (https://docs.docker.com/articles/networking/)
- https://github.com/cajus/uif (https://github.com/cajus/uif)

Date September 18, 2015 (2015-09-18T11:41:00+02:00) **By** @jeekajoo (./author/jeekajoo.html) **Category** docker (./category/docker.html) **Tags** docker (./tag/docker.html) container (./tag/container.html) iptables (./tag/iptables.html) security (./tag/security.html) firewall (./tag/firewall.html) network (./tag/network.html)