Understand container communication

Estimated reading time: 5 minutes

The information in this section explains container communication within the Docker default bridge. This is a `bridge` network named `bridge` created automatically when you install Docker.

Note: The Docker networks feature allows you to create user-defined networks in addition to the default bridge network.

Communicating to the outside world

Whether a container can talk to the world is governed by two factors. The first factor is whether the host machine is forwarding its IP packets. The second is whether the host's `iptables` allow this particular connection.

IP packet forwarding is governed by the `ip_forward` system parameter. Packets can only pass between containers if this parameter is `1`. Usually you will simply leave the Docker server at its default setting `--ip-forward=true` and Docker will go set `ip_forward` to `1` for you when the server starts up. If you set `--ip-forward=false` and your system's kernel has it enabled, the `--ip-forward=false` option has no effect. To check the setting on your kernel or to turn it on manually:

```
$ sysctl net.ipv4.conf.all.forwarding

net.ipv4.conf.all.forwarding = 0

$ sysctl net.ipv4.conf.all.forwarding=1

$ sysctl net.ipv4.conf.all.forwarding
```

```
net.ipv4.conf.all.forwarding = 1
```

> **Note**: this setting does not affect containers that use the host network stack (`--network=host`).

Many using Docker will want `ip_forward` to be on, to at least make communication *possible* between containers and the wider world. May also be needed for inter-container communication if you are in a multiple bridge setup.

Docker will never make changes to your system `iptables` rules if you set `--iptables=false` when the daemon starts. Otherwise the Docker server will append forwarding rules to the `DOCKER` filter chain.

Docker will not delete or modify any pre-existing rules from the `DOCKER` filter chain. This allows the user to create in advance any rules required to further restrict access to the containers.

Docker's forward rules permit all external source IPs by default. To allow only a specific IP or network to access the containers, insert a negated rule at the top of the `DOCKER` filter chain. For example, to restrict external access such that *only* source IP 8.8.8.8 can access the containers, the following rule could be added:

```
$ iptables -I DOCKER -i ext_if ! -s 8.8.8.8 -j DROP
```

where *ext_if* is the name of the interface providing external connectivity to the host.

Communication between containers

Whether two containers can communicate is governed, at the operating system level, by two factors.

- Does the network topology even connect the containers' network interfaces? By default Docker will attach all containers to a single `docker0` bridge, providing a path for packets to travel between them. See the later sections of this document for other possible topologies.

- Do your `iptables` allow this particular connection? Docker will never make changes to your system `iptables` rules if you set `--iptables=false` when the daemon starts. Otherwise the Docker server will add a default rule to the `FORWARD` chain with a blanket `ACCEPT` policy if you retain the default `--icc=true`, or else will set the policy to `DROP` if `--icc=false`.

It is a strategic question whether to leave `--icc=true` or change it to `--icc=false` so that `iptables` will protect other containers – and the main host – from having arbitrary ports probed or accessed by a container that gets compromised.

If you choose the most secure setting of `--icc=false`, then how can containers communicate in those cases where you *want* them to provide each other services? The answer is the `--link=CONTAINER_NAME_or_ID:ALIAS` option, which was mentioned in the previous section because of its effect upon name services. If the Docker daemon is running with both `--icc=false` and `--iptables=true` then, when it sees `docker run` invoked with the `--link=` option, the Docker server will insert a pair of `iptables` `ACCEPT` rules so that the new container can connect to the ports exposed by the other container – the ports that it mentioned in the `EXPOSE` lines of its `Dockerfile`.

> **Note**: The value `CONTAINER_NAME` in `--link=` must either be an auto-assigned Docker name like `stupefied_pare` or else the name you assigned with `--name=` when you ran `docker run`. It cannot be a hostname, which Docker will not recognize in the context of the `--link=` option.

You can run the `iptables` command on your Docker host to see whether the `FORWARD` chain has a default policy of `ACCEPT` or `DROP`:

```
# When --icc=false, you should see a DROP rule:

$ sudo iptables -L -n

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

# When a --link= has been created under --icc=false,
# you should see port-specific ACCEPT rules overriding
# the subsequent DROP policy for all other packets:

$ sudo iptables -L -n

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

Chain DOCKER (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  172.17.0.2           172.17.0.3           tcp spt:
ACCEPT     tcp  --  172.17.0.3           172.17.0.2           tcp dpt:
```

> **Note**: Docker is careful that its host-wide `iptables` rules fully expose containers to each other's raw IP addresses, so connections from one container to another should always appear to be originating from the first container's own IP address.

Feedback? Suggestions? Can't find something in the docs?
Edit this page ● Request docs changes ● Get support
Rate this page: