Networking overview

Container networking refers to the ability for containers to connect to and communicate with each other, or to non-Docker workloads.

Containers have networking enabled by default, and they can make outgoing connections. A container has no information about what kind of network it's attached to, or whether their peers are also Docker workloads or not. A container only sees a network interface with an IP address, a gateway, a routing table, DNS services, and other networking details. That is, unless the container uses the `none` network driver.

This page describes networking from the point of view of the container, and the concepts around container networking. This page doesn't describe OS-specific details about how Docker networks work. For information about how Docker manipulates `iptables` rules on Linux, see Packet filtering and firewalls.

User-defined networks

You can create custom, user-defined networks, and connect multiple containers to the same network. Once connected to a user-defined network, containers can communicate with each other using container IP addresses or container names.

The following example creates a network using the `bridge` network driver and running a container in the created network:

```
$ docker network create -d bridge my-net
$ docker run --network=my-net -itd --name=container3 busybox
```

Drivers

The following network drivers are available by default, and provide core networking functionality:

Driver	Description
`bridge`	The default network driver.
`host`	Remove network isolation between the container and the Docker host.
`none`	Completely isolate a container from the host and other containers.
`overlay`	Overlay networks connect multiple Docker daemons together.
`ipvlan`	IPvlan networks provide full control over both IPv4 and IPv6 addressing.
`macvlan`	Assign a MAC address to a container.

For more information about the different drivers, see Network drivers overview.

Container networks

In addition to user-defined networks, you can attach a container to another container's networking stack directly, using the `--network container:<name|id>` flag format.

The following flags aren't supported for containers using the `container:` networking mode:

- `--add-host`
- `--hostname`
- `--dns`
- `--dns-search`
- `--dns-option`
- `--mac-address`
- `--publish`
- `--publish-all`
- `--expose`

The following example runs a Redis container, with Redis binding to `localhost`, then running the `redis-cli` command and connecting to the Redis server over the `localhost` interface.

```
$ docker run -d --name redis example/redis --bind 127.0.0.1
$ docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1
```

Published ports

By default, when you create or run a container using `docker create` or `docker run`, the container doesn't expose any of its ports to the outside world. Use the `--publish` or `-p` flag to make a port available to services outside of Docker. This creates a firewall rule in the host, mapping a container port to a port on the Docker host to the outside world. Here are some examples:

Flag value	Description
`-p 8080:80`	Map port `8080` on the Docker host to TCP port `80` in the container.
`-p 192.168.1.100:8080:80`	Map port `8080` on the Docker host IP `192.168.1.100` to TCP port `80` in the container.
`-p 8080:80/udp`	Map port `8080` on the Docker host to UDP port `80` in the container.
`-p 8080:80/tcp -p 8080:80/udp`	Map TCP port `8080` on the Docker host to TCP port `80` in the container, and map UDP port `8080` on the Docker host to UDP port `80` in the container.

> **Important**
>
> Publishing container ports is insecure by default. Meaning, when you publish a container's ports it becomes available not only to the Docker host, but to the outside world as well.
>
> If you include the localhost IP address (`127.0.0.1`) with the publish flag, only the Docker host can access the published container port.
>
> ```
> $ docker run -p 127.0.0.1:8080:80 nginx
> ```
>
> > **Warning**

> Hosts within the same L2 segment (for example, hosts connected to the same network switch) can reach ports published to localhost. For more information, see moby/moby#45610

If you want to make a container accessible to other containers, it isn't necessary to publish the container's ports. You can enable inter-container communication by connecting the containers to the same network, usually a bridge network.

IP address and hostname

By default, the container gets an IP address for every Docker network it attaches to. A container receives an IP address out of the IP subnet of the network. The Docker daemon performs dynamic subnetting and IP address allocation for containers. Each network also has a default subnet mask and gateway.

You can connect a running container to multiple networks, either by passing the `--network` flag multiple times when creating the container, or using the `docker network connect` command for already running containers. In both cases, you can use the `--ip` or `--ip6` flags to specify the container's IP address on that particular network.

In the same way, a container's hostname defaults to be the container's ID in Docker. You can override the hostname using `--hostname`. When connecting to an existing network using `docker network connect`, you can use the `--alias` flag to specify an additional network alias for the container on that network.

DNS services

Containers use the same DNS servers as the host by default, but you can override this with `--dns`.

By default, containers inherit the DNS settings as defined in the `/etc/resolv.conf` configuration file. Containers that attach to the default `bridge` network receive a copy of this file. Containers that attach to a custom network use Docker's embedded DNS server. The embedded DNS server forwards external DNS lookups to the DNS servers configured on the host.

You can configure DNS resolution on a per-container basis, using flags for the `docker run` or `docker create` command used to start the container. The following table describes the available `docker run` flags related to DNS configuration.

Flag	Description
`--dns`	The IP address of a DNS server. To specify multiple DNS servers, use multiple `--dns` flags. If the container can't reach any of the IP addresses you specify, it uses Google's public DNS server at `8.8.8.8`. This allows containers to resolve internet domains.
`--dns-search`	A DNS search domain to search non-fully qualified hostnames. To specify multiple DNS search prefixes, use multiple `--dns-search` flags.
`--dns-opt`	A key-value pair representing a DNS option and its value. See your operating system's documentation for `resolv.conf` for valid options.
`--hostname`	The hostname a container uses for itself. Defaults to the container's ID if not specified.

Nameservers with IPv6 addresses

If the `/etc/resolv.conf` file on the host system contains one or more nameserver entries with an IPv6 address, those nameserver entries get copied over to `/etc/resolv.conf` in containers that you run.

For containers using musl libc (in other words, Alpine Linux), this results in a race condition for hostname lookup. As a result, hostname resolution might sporadically fail if the external IPv6 DNS server wins the race condition against the embedded DNS server.

It's rare that the external DNS server is faster than the embedded one. But things like garbage collection, or large numbers of concurrent DNS requests, can sometimes result in a round trip to the external server being faster than local resolution.

Custom hosts

Your container will have lines in `/etc/hosts` which define the hostname of the container itself, as well as `localhost` and a few other common things. Custom hosts, defined in `/etc/hosts` on the host machine, aren't inherited by containers. To pass additional hosts into a container, refer to add entries to container hosts file in the `docker run` reference documentation.

Proxy server

If your container needs to use a proxy server, see Use a proxy server.