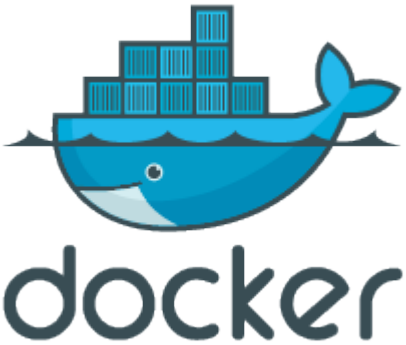
Overview - How does Docker Networks work?



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 its 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. It doesn't cover OS-specific details about how Docker networks work. For more 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, containers can communicate with each other using container IP addresses or container names.

```
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`	Removes network isolation between the container and the Docker host.
`none`	Completely isolates a container from the host and other containers.
`overlay`	Connects multiple Docker daemons together.
`ipvlan`	Provides full control over IPv4 and IPv6 addressing.
`macvlan`	Assigns a MAC address to a container.

> For more information, take a deepdive into the different Drivers at this Post.

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 example demonstrates running a Redis container with Redis binding to localhost, then running the redis-cli command and connecting to the Redis server:

```
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, containers on bridge networks don’t expose ports to the outside world. Use the `--publish` or `-p` flag to make a port available externally. Examples:

Flag Value	Description
`-p 8080:80`	Maps port 8080 on the Docker host to TCP port 80 in the container.
`-p 192.168.1.100:8080:80`	Maps port 8080 on host IP 192.168.1.100 to TCP port 80 in the container.
`-p 8080:80/udp`	Maps UDP port 8080 on the host to UDP port 80 in the container.

Important: Publishing container ports is insecure by default. To restrict access, bind ports to `localhost` or specific IP addresses.

IP Address and Hostname

Containers receive an IP address for every network they attach to. The Docker daemon dynamically assigns these IPs based on the network’s subnet. You can specify IP addresses manually using the `--ip` or `--ip6` flags.

By default, a container’s hostname is its ID. You can override this using `--hostname`.
For additional network aliases, use the `--alias` flag when connecting a container to a network.

For more details, see the official documentation.

DNS Services

By default, containers inherit DNS settings from the host. You can override these settings using the following flags:

Flag	Description
`--dns`	The IP address of a DNS server.
`--dns-search`	A DNS search domain for non-fully qualified hostnames.
`--dns-opt`	Key-value pairs for DNS options.

For more details, see the official documentation.

Revision #8
Created 18 September 2024 08:34:22 by aeoneros
Updated 15 January 2025 15:36:16 by aeoneros