Docker Networking

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Objectives

1. Understanding Docker Networking in detail.
2. Understanding different network Drivers in detail
    - bridge
    - host
    - overlay
    - ipvlan
    - macvlan

Docker Networking

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

- 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.

Publishing 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 TCP port 80 in the container to port 8080 on the Docker host.
-p 192.168.1.100:8080:80	Map TCP port 80 in the container to port 8080 on the Docker host for connections to host IP 192.168.1.100.
-p 8080:80/udp	Map UDP port 80 in the container to port 8080 on the Docker host.
-p 8080:80/tcp -p 8080:80/udp	Map TCP port 80 in the container to TCP port 8080 on the Docker host, and map UDP port 80 in the container to UDP port 8080 on the Docker host.

- **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

IP Address and Host Name

- 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.**

- When a container starts, it can only **attach to a single network**, using the **--network flag**.

- When you connect an existing container/running container to a different network using **docker network connect**, you can use the **--ip** or **--ip6** flags on that command to specify the **container's IP address on the additional 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.

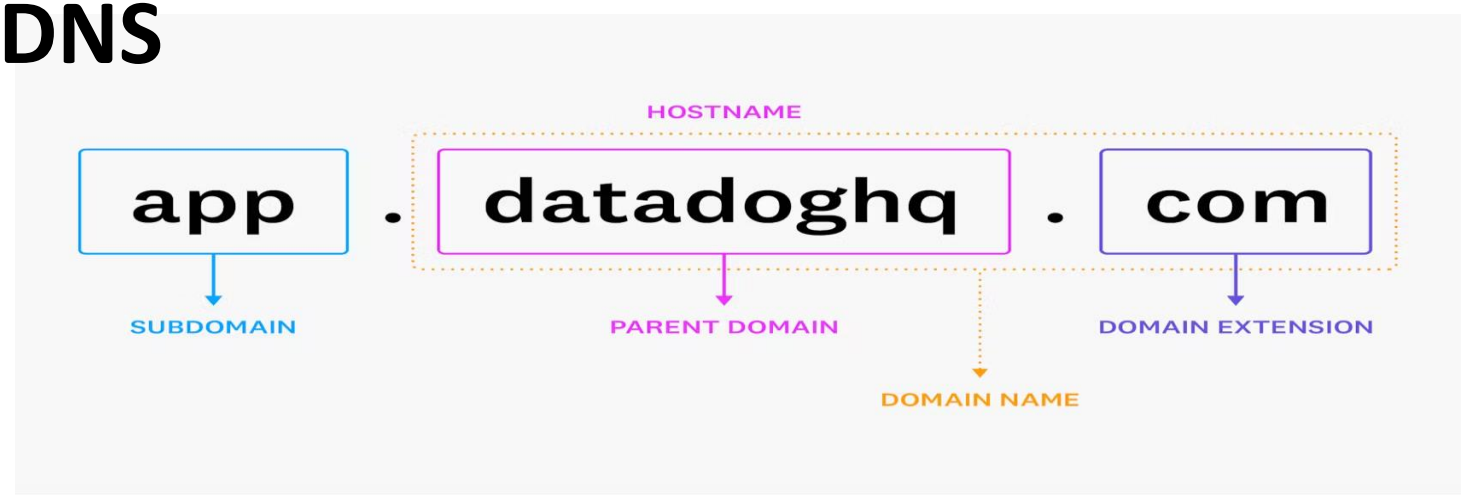
DNS services

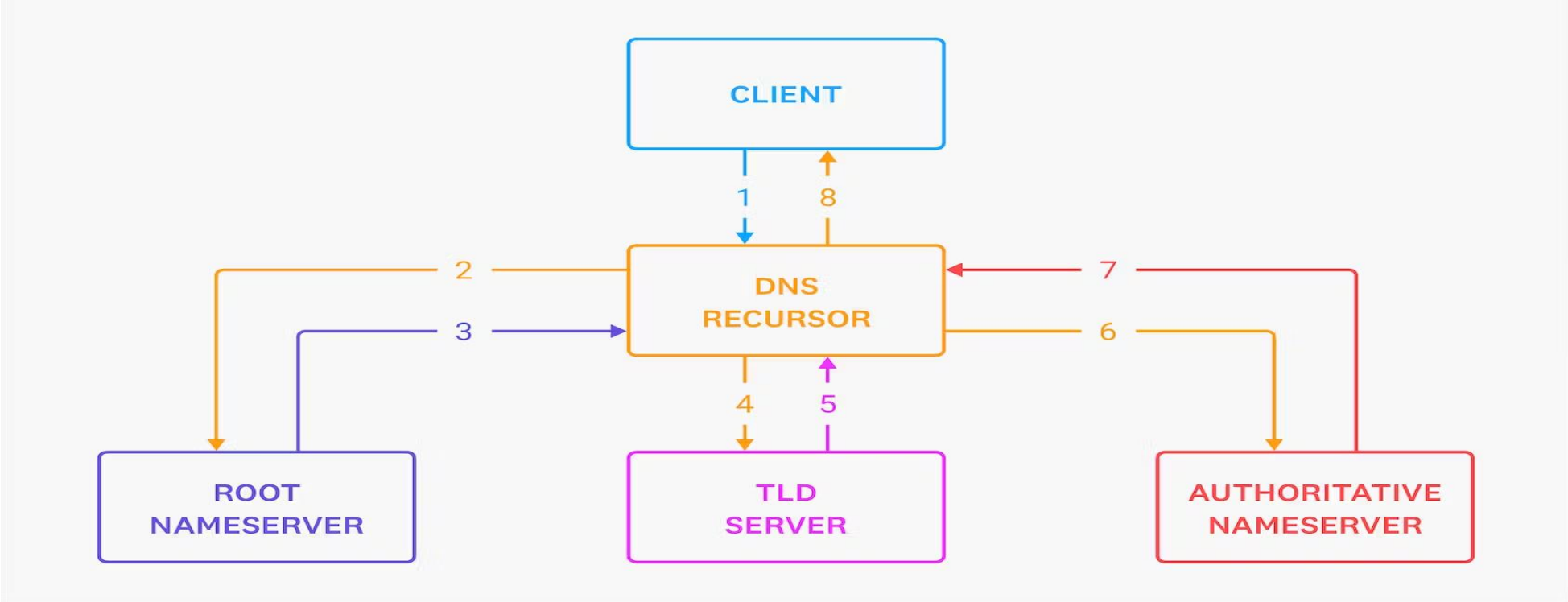
- By default, **containers inherit the DNS settings of the host**, as defined in the **/etc/resolv.conf configuration file**.
- Containers that attach to the **default bridge network** receive a **copy of this file**.
- Containers that attach to a **custom network use Docker's embedded DNS server**. 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 table on next slide describes the available **docker run flags related to DNS configuration**.

DNS services

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.

DNS


HOSTNAME
app
.
datadoghq
.
com
SUBDOMAIN
PARENT DOMAIN
DOMAIN NAME
DOMAIN EXTENSION


CLIENT
1
8
2
3
DNS
RECURSOR
7
6
4
5
ROOT
NAMESERVER
TLD
SERVER
AUTHORITATIVE
NAMESERVER

Network Drivers

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

- **bridge:** The **default network driver.** If you don't specify a driver, this is the type of network you are creating. Bridge networks are commonly used **when your application runs in a container that needs to communicate with other containers on the same host.**

- **host:** **Removes network isolation between the container and the Docker host**, and use the host's networking directly.

- **overlay:** Overlay networks **connect multiple Docker daemons together and enable Swarm services** and **containers to communicate across nodes.** This strategy removes the need to do OS-level routing.

- **ipvlan:** IPvlan networks give users **total control over both IPv4 and IPv6 addressing**. The VLAN driver builds on top of that in giving operators complete control of **layer 2 VLAN tagging** and even **IPvlan L3 routing** for users interested in underlay network integration.

Network Drivers

macvlan: Macvlan networks allow you to **assign a MAC address to a container**, **making it appear as a physical device on your network**. The Docker **daemon routes traffic to containers by their MAC addresses**. Using the macvlan driver is sometimes the best choice when dealing **with legacy applications that expect to be directly connected to the physical network,** rather than routed through the Docker host's network stack.

none: **Completely isolate a container from the host and other containers**. none is not available for Swarm services.

Network plugins: You can install and **use third-party network plugins** with Docker.

Network Drivers Use Cases Summary

•The default bridge network is good for **running containers that don't require special networking capabilities**.

•User-defined bridge networks enable **containers on the same Docker host to communicate with each other**. A user-defined network typically defines an isolated network for **multiple containers belonging to a common project or component.**

•Host network shares **the host's network with the container**. When you use this driver, the container's network isn't isolated from the host.

•Overlay networks are best when you **need containers running on different Docker hosts to communicate**, or when multiple applications work together using **Swarm services**.

•Macvlan networks are best **when you are migrating from a VM setup or need your containers to look like physical hosts on your network**, each with a unique MAC address.

•IPvlan is similar to Macvlan, but doesn't assign unique MAC addresses to containers. Consider using IPvlan when there's a **restriction on the number of MAC addresses** that can be assigned to a network interface or port.

•Third-party network plugins allow you to integrate Docker with specialized network stacks.

Bridge Network Driver

- In terms of networking, a bridge network is a **Link Layer device which forwards traffic between network segments**.
- A bridge can be a **hardware device or a software device** running within a host machine's kernel.
- In terms of Docker, a bridge network uses a **software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network.**
- The **Docker bridge driver automatically installs rules in the host machine** so that containers on different bridge networks cannot communicate directly with each other.
- Bridge networks **apply to containers running on the same Docker daemon host**. For communication among containers running on different Docker daemon hosts, you can either **manage routing at the OS level, or you can use an overlay network.**
- When you start Docker, a **default bridge network (also called bridge) is created automatically**, and newly-started containers connect to it unless otherwise specified. You can also create **user-defined custom bridge networks.** User-defined bridge networks are superior to the default bridge network.

Differences between user-defined bridges and the default bridge

- User-defined bridges provide **automatic DNS resolution between containers**.
- User-defined bridges **provide better isolation.**
- Each user-defined network **creates a configurable bridge.**
- Linked containers on the **default bridge network share environment variables.**
- You can use **swarm services instead of standalone containers**, and take advantage of shared secrets and configs.

Manage a user-defined bridge

Use the **docker network create command** to create a user-defined bridge network.

docker network create my-net

You can specify the **subnet, the IP address range, the gateway, and other options**.

Use the **docker network rm** command to remove a user-defined bridge network. If containers are currently connected to the network, disconnect them first.

docker network rm my-net

Connect a container to a user-defined bridge

When you create a new container, **you can specify one or more --network flags.** This example connects a Nginx container to the my-net network. It also publishes port 80 in the container to port 8080 on the Docker host, so external clients can access that port. Any other container connected to the my-net network has access to all ports on the my-nginx container, and vice versa.

***docker create --name my-nginx ***
***--network my-net ***
***--publish 8080:80 ***
nginx:latest

To connect a running container to an existing user-defined bridge, use the **docker network connect** command. The following command connects an already-running my-nginx container to an already-existing my-net network:

docker network connect my-net my-nginx

Manage a user-defined bridge

Disconnect a container from a user-defined bridge

To disconnect a running container from a user-defined bridge, use the **docker network disconnect** command. The following command disconnects the my-nginx container from the my-net network.

docker network disconnect my-net my-nginx

Connect a container to the default bridge network

If you **do not specify a network using the --network flag**, and you do specify a network driver, **your container is connected to the default bridge network by default.** Containers connected to the default bridge network can communicate, but only by IP address, unless they are linked using the legacy --link flag.

To configure the default bridge network, you specify options in **daemon.json.** Here is an example **daemon.json** with several options specified. Only specify the settings you need to customize.

```
{
"bip": "192.168.1.1/24",
"fixed-cidr": "192.168.1.0/25",
"fixed-cidr-v6": "2001:db8::/64",
"mtu": 1500,
"default-gateway": "192.168.1.254",
"default-gateway-v6": "2001:db8:abcd::89",
"dns": ["10.20.1.2","10.20.1.3"]
}
```

Overlay network driver

- The overlay network driver creates a **distributed network among multiple Docker daemon hosts**. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate securely when encryption is enabled.

- Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container.

- When you initialize a **swarm or join a Docker host to an existing swarm, two new networks are created** on that Docker host:
    - **an overlay network called ingress**, which handles the control and data traffic related to swarm services. When you create a swarm service and do not connect it to a user-defined overlay network, it connects to the ingress network by default.
    - **a bridge network called docker_gwbridge**, which connects the individual Docker daemon to the other daemons participating in the swarm.
- You **can create user-defined overlay networks using docker network create**, in the same way that you can create user-defined bridge networks.
- **Services or containers can be connected to more than one network at a time**. Services or containers can only communicate across networks they are each connected to.
- Although you can connect **both swarm services and standalone containers to an overlay network**, the default behaviors and configuration concerns are different.

Overlay network driver

- You can create user-defined overlay networks using docker network create, in the same way that you can create user-defined bridge networks.

- Services or containers can be connected to more than one network at a time. Services or containers can only communicate across networks they are each connected to.

- Although you can connect both swarm services and standalone containers to an overlay network, the default behaviors and configuration concerns are different.

Host Network Driver

- If you use the host network mode for a container, that container's network stack is not isolated from the Docker host (the container shares the host's networking namespace), and the container does not get its own IP-address allocated.
- For instance, if you run a container which binds to port 80 and you use host networking, the container's application is available on port 80 on the host's IP address.

Note

- Given that the container does not have its own IP-address when using host mode networking, port-mapping does not take effect, and the -p, --publish, -P, and --publish-all option are ignored, producing a warning instead:

- Host mode networking can be useful to optimize performance, and in situations where a container needs to handle a large range of ports, as it does not require network address translation (NAT), and no "userland-proxy" is created for each port.

- The host networking driver only works on Linux hosts, and is not supported on Docker Desktop for Mac, Docker Desktop for Windows, or Docker EE for Windows Server.

- You can also use a host network for a swarm service, by passing --network host to the docker service create command. In this case, control traffic (traffic related to managing the swarm and the service) is still sent across an overlay network, but the individual swarm service containers send data using the Docker daemon's host network and ports. This creates some extra limitations. For instance, if a service container binds to port 80, only one service container can run on a given swarm node.