Docker networking:

There are 3 major components that consists of docker networking

- Container network model(CNM)
The first one is the container network model, it is the design specification and it outlines the fundamental building blocks of Docker network.

- The libnetwork implements CNM
The second component is the libnetwork, it is the real-world implementation of the CNM and docker uses it for connecting containers. Libnetwork is also responsible for service discovery, ingress-based container load balancing, and the network and management control plane functionality. Libnetwork uses a system of drivers.

- Driver extends the model by network topologies
Driver extends the model by implementing specific network topologies.

Network drivers

1. **bridge:** It is a default network. It is a link layer device, which forwards traffic between network segments. It uses a software bridge. It only works on Linux.

2. **host:** Remove network isolation between the container and the Docker host, and use the host's networking directly.
3. **overlay:** Used on connecting containers in multiple hosts
4. **macvlan:** Allows you to assign a MAC address to a container and this gives it the appearance of being a physical device on your network. E.g an application that monitors networking traffic, and those applications are expected to be physically connected to a network.
5. **none:** To disable the network. Used in conjunction with a custom network driver. Cannot be used in swarm service.
6. **Network plugin**

Network commands

List networks
docker network ls

Getting detailed info on a network
docker network inspect <NAME>

Create a network

```
docker network create <NAME>
```

Removing a network

```
docker network rm <NAME>
```

Remove all unused networks

```
docker network prune
```

Adding a container to a network

```
docker network connect <NETWORK> <CONTAINER>
```

Removing a container from a network

```
docker network disconnect <NETWORK> <CONTAINER>
```

Example

```
docker container run -d --name network-test -p 8081:80 nginx
docker network create br01
docker network connect br01 network-test
docker network inspect network-test
docker network disconnect br01 network-test
```

Deep dive

Networking containers

Create a network with a Subnet and Gateway

```
docker network create --subnet <SUBNET> --gateway <GATEWAY> <NAME>
docker network create --subnet <SUBNET> --gateway <GATEWAY> --ip-range=<IP_RANGE>
--driver=<DRIVER>
```

Example

```
docker network create --subnet 10.1.0.0/24 --gateway 10.1.0.1 br02
docker network ls
docker network inspect br02
docker network create --subnet 10.1.0.0/16 --gateway 10.1.0.1 --ip-range=10.1.4.0/24
--driver=bridge --label=host4network br04
docker network ls
```

Removing a network

```
docker network rm <NAME>
```

Adding a container to a network

```
docker container run -name <NAME> -it --network <NETWORK> <IMAGE> <CMD>
```

Assigning an IP to a container

```
docker container run --name <NAME> -it --network <NETWORK> --ip <IP> <IMAGE> <CMD>
```

Adding a container to a network

```
docker container connect <NETWORK> <CONTAINER>
```

Example

```
docker container run --name network-test -it --network br01 centos /bin/bash
docker container run -d --name network-test --ip 10.1.4.102 --network br02 nginx
docker container inspect network-test | grep IPAddr
docker network create -d bridge --internal localhost
docker container run -d --name test_mysql -e MYSQL_ROOT_PASSWORD=P4sSw0rd0
--network localhost mysql:5.7
docker container run -it --name ping-mysql --network bridge centos
docker container connect localhost ping-mysql
docker container start -ia ping-mysql
docker container run -d --name private-network -p 8081:80 --network localhost nginx
curl localhost:8081
docker inspect private-network
curl 182.28.0.3
```

Networking in docker

https://docs.docker.com/network/

Container Volumes

Persistent storage for volatile containers

- Containers are volatile in nature because they are disposable, making changes in container(adding packages, configurations) are done through image
- The data doesn't persist when that container no longer exists and it can be difficult to get the data of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.
- Removing container deletes the data
- In the case of a stateful container such as mysql, stores database and reads from database, in such a case we have container volumes.

Docker has two options for containers to store files in the host machine
so that the files are persisted even after the container stops

Volumes
Bind Mounts

Use of Volumes

1. Decoupling container from storage
2. Share volume (storage/data) among different containers
3. Attach volume to container
4. On deleting container volume does not delete

By default all files created inside a container are stored on a writable container layer

Volumes and BIND mounts

- Volumes are stored in a part of the host filesystem which is managed by Docker
- Non-Docker processes should not modify this part of the filesystem
- Bind mounts may be stored anywhere on the host system
- Non-Docker processes on the Docker host or a Docker container can modify them at any time
- In Bind Mounts, the file or directory is referenced by its full path on the host machine.
- Volumes are the best way to persist data in Docker
- volumes are managed by Docker and are isolated from the core functionality of the host machine
- A given volume can be mounted into multiple containers simultaneously.
- When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using docker volume prune.
- When you mount a volume, it may be named or anonymous.
- Anonymous volumes are not given an explicit name when they are first mounted into a container
- Volumes also support the use of volume drivers, which allow you to store your data on remote hosts or cloud providers, among other possibilities.

docker volume {options} *volumeName*
(ls, create, inspect, prune, rm)

Volume commands

List all Docker volume commands

```
docker volume -h
```

List all volumes on the host

```
docker volume ls
```

Creating volumes

```
docker volume create <NAME>
```

Inspecting a volume

```
docker volume inspect <NAME>
```

Deleting volume

```
docker volume rm <NAME>
```

Removing all unused volumes

```
docker volume prune
```

Example

```
docker volume create test-vol1
docker volume create test-vol2
docker volume inspect test-vol1
```

Bind mounts

Using the mount flag

```
docker container run -d --name <NAME> --mount
type=bind,source=<SOURCE>,target=<TARGET> <IMAGE>
```

Using the volume flag

```
docker container run -d --name <NAME> -v <source>:<TARGET> <IMAGE>
```

Example

```
mkdir target
docker container run -d --name nginx-bind-mnt1 --mount
type=bind,source="$(pwd)"/target,target=/app nginx

docker container run -d --name nginx-bind-mnt2 -v "$(pwd)"/target2:/app nginx
```

Volumes for storage

Create a new volume for a Nginx container

```
docker volume create html-volume
```

Using the mount flag

```
docker container run -d --name <NAME> --mount
type=volume,source=<SOURCE>,target=<TARGET> <IMAGE>
```

Creating volume using the volume flag

```
docker container run -d --name <NAME> -v <VOLUME-NMAE>:<TARGET>
<IMAGE>
```

Examples

```
docker volume create html-volume
docker container run -d --name nginx-vol1 --mount
type=volume,source=html-volume,target=/usr/share/nginx/html/ nginx

docker container run -d --name nginx-vol2 -v html-vol:/usr/share/nginx/html nginx
docker container run -d --name nginx-vol3 --mount
source=html-volume,target=/usr/share/nignx/html,readonly nginx


docker volume create devOpsvol1
docker run --name jenkins -p 8080:8080 -p 50000:50000 --restart=on-failure -v
devOpsvol1:/var/jenkins_home jenkins/jenkins:lts-jdk11
```

```
docker run --name jenkins1 -p 8081:8080 -p 50001:50000 --restart=on-failure -v
devOpsvol1:/var/jenkins_home jenkins/jenkins:lts-jdk11
```

```
docker run --name jenkins2 -p 8082:8080 -p 50002:50000 --restart=on-failure -v
/var/jenkins_home jenkins/jenkins:lts-jdk11
```

```
 docker run --name jenkins3 -p 8083:8080 -p 50003:50000 --restart=on-failure -v
/opt/docker/vol2:/var/jenkins_home jenkins/jenkins:lts-jdk11
```

Ref:
https://docs.docker.com/storage/