Docker

A platform for building, running and sharing the application.

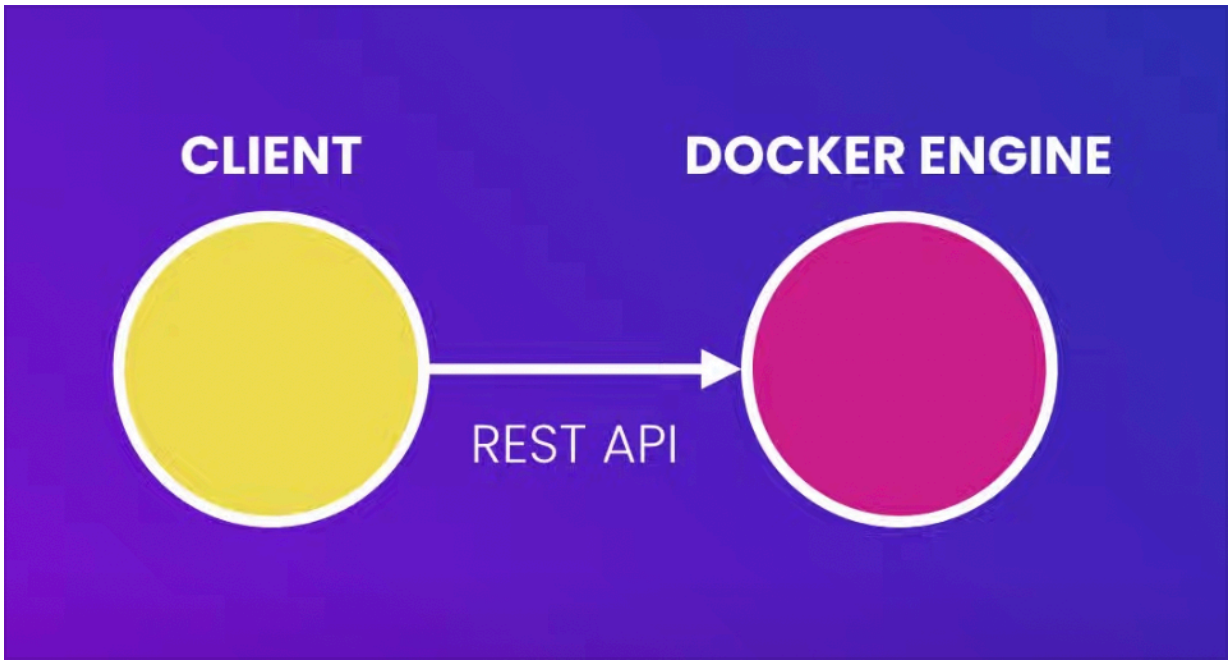
Reasons:
- One or more file missing
- Software version mismatch
- Different configuration settings

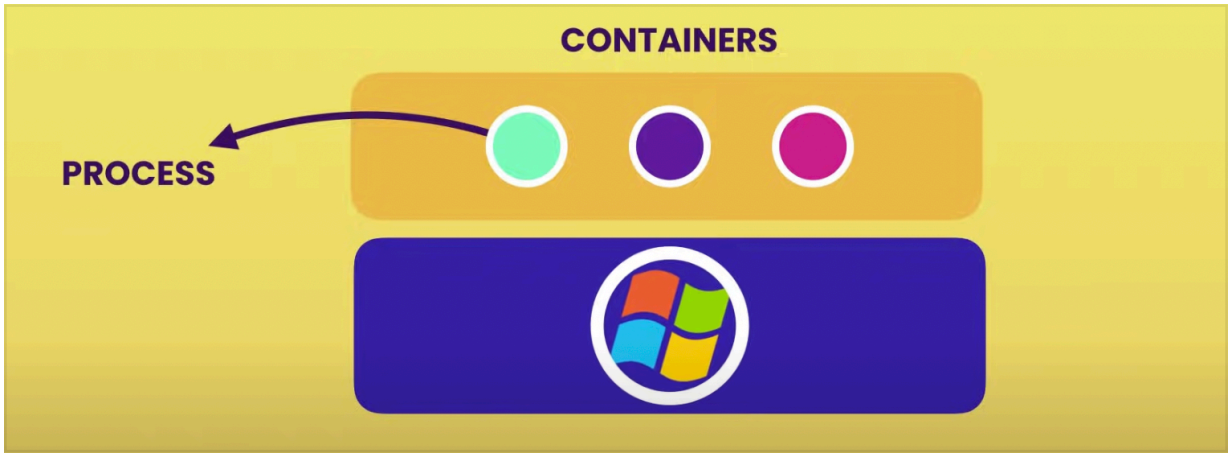
Virtual Machine
- An abstraction of machine (physical hardware)

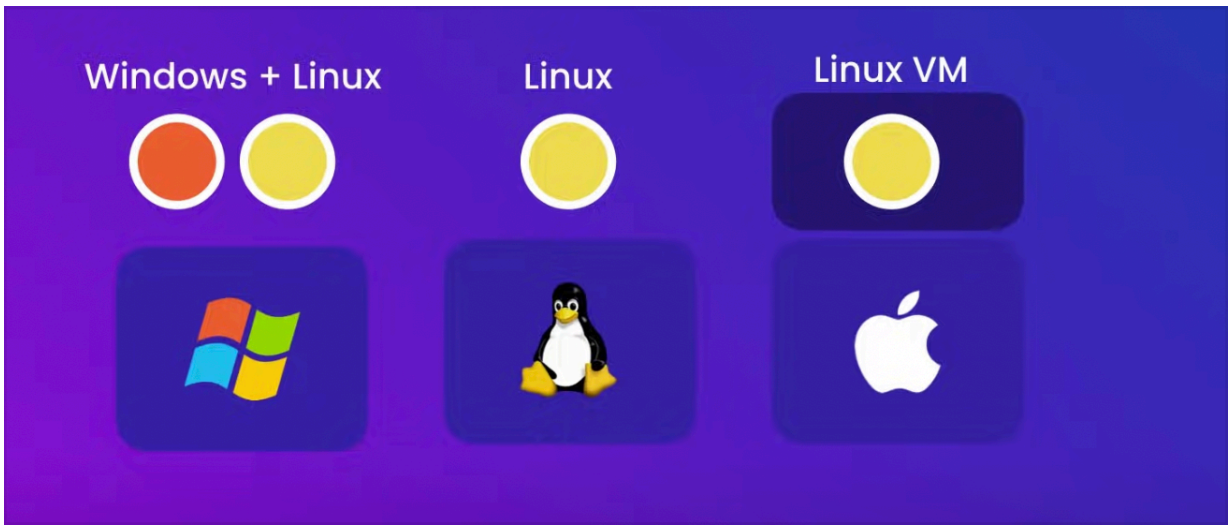
VM problems:
- Each VM needs a full blown OS.
- Slow to start
- Resource internsive (each VM needs actual physical hardware resources like CPU, RAM, HDD)
-

Container
- An isolated environment for running an application.
- Allow running multiple apps in isolation
- Are lightweight
- Use OS of the host (Each container shares the kernel of the host)
  Note: kernel manages applications and hardware resources
- Start quickly
- Needs less hardware resources
- Because of less H/W resources, we can run multiple container side by side in single system.

Docker architecture







Windows container use windows kernel same as Mac and linux container

usage its own kernel to build application.

Note: In windows 10 there is two kernel inbuilt with OS, windows as well as linux kernel that means we can run linux applications in windows and vice versa.

Development workflow

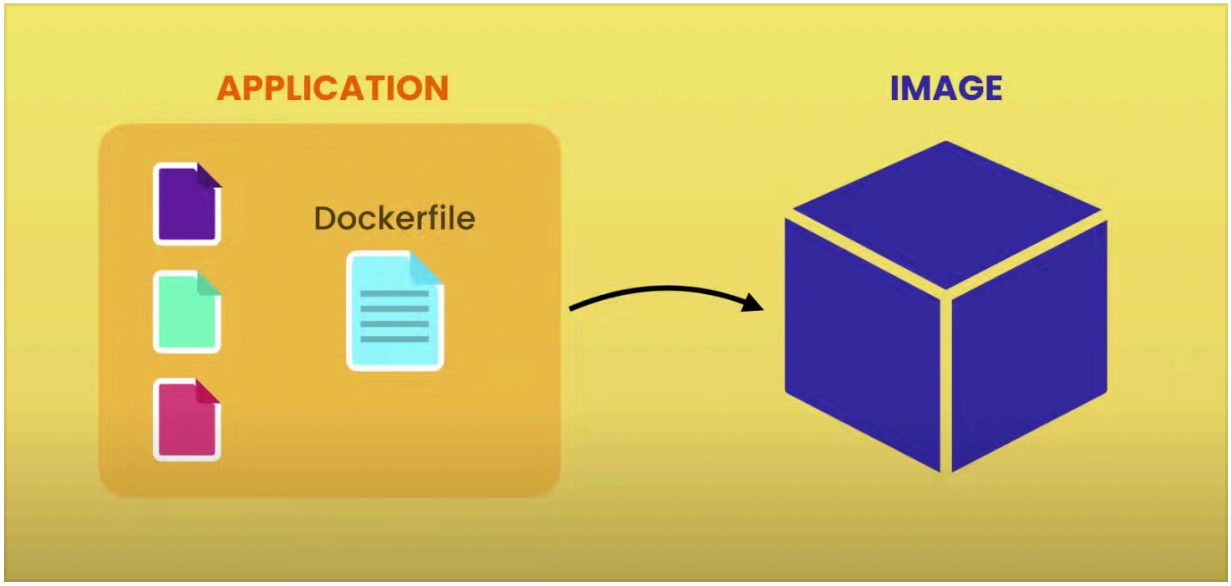
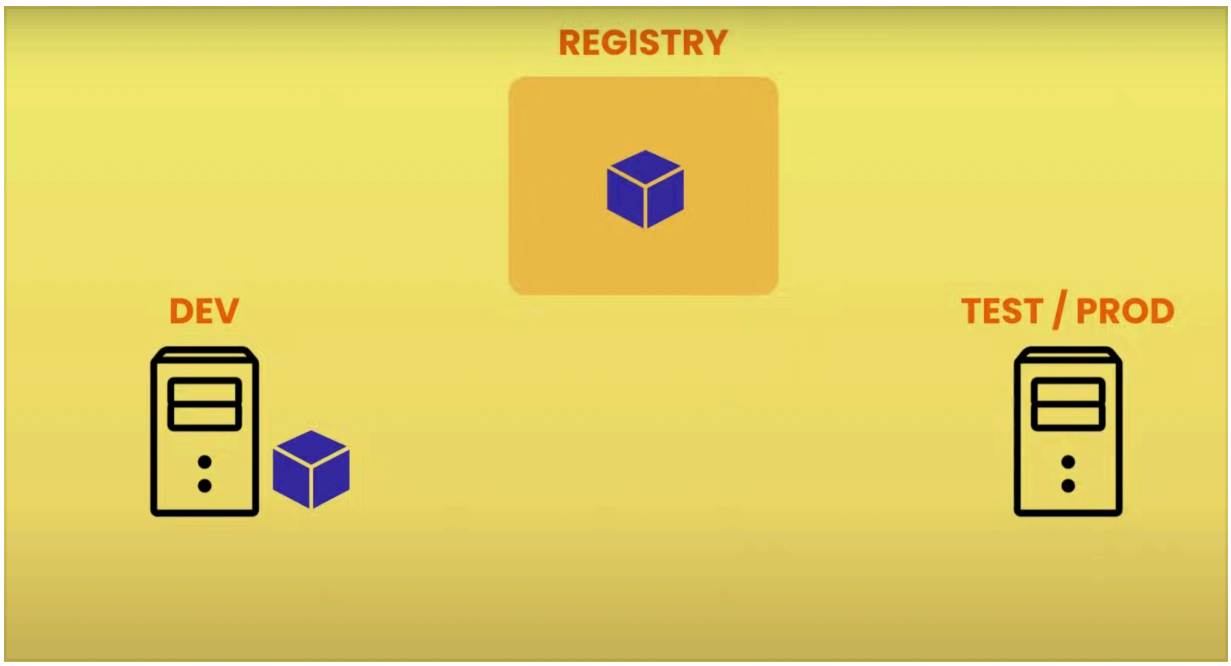


Image:

- A cut down OS
- A runtime environment (eg. Node)
- Application files
- Third party libraries
- Environment variables

- Image loaded into container and container running into the system.
- Container is a process and it has its own file system which is provided by the image

In a project one docker file is loaded and the docker file has all the application data like java, Nodejs, reactjs, etc. and that docker file attached to the image, so image has all the data. Then, container makes from the multiple images, and the container has its own file system that is provided by the images. This is why we run our application locally in our development machine.



Then we send our images into docker hub which present inside the registry

just like GitHub to git. Registry is a storage where all the docker images are present that anyone can use. And we fetch that images in other pc, and test it if it's working. We put it images virtually and run it anywhere. This is the beauty of the docker.

Docker action:

mkdir hello-docker

cd hello-docker

For example:
Instructions

- Start with an OS
- Install node
- Copy app file
- Run node app.js

Dockerhub is the registry that has the multiple server like node that's installed in the multiple linuxes. And for example, if we building the node, if the building the java application, that is already presents in the server. And if its not in the server then we can also download into those GitHub registry which has the multiple linuxes.

So, we put instructions into the docker file.
select the server

- FROM node:alpine (alpine is the linux server that is available in docker hub registry.)
  then we copied local file to the app directory to the image which is file system and over there we created directory called app.
- COPY . /app
  when we use workdir then all the beloved instruction know that we are in the app directory otherwise we forcelly put the CMD app/node.js in every instruction belowed.
- WORKDIR /app
- CMD node.js

We run,
docker build -t first_docker .

- Dot defined the current directory of the Dockerfile

docker image ls

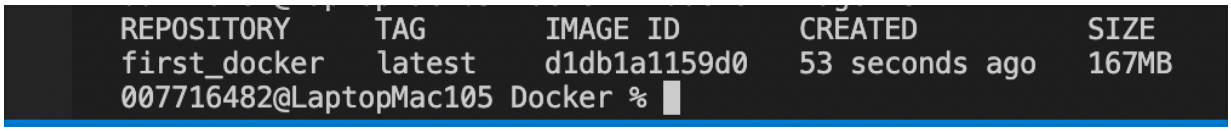


Image store virtually in dockerhub.
TAG in above image means image contain different version of our application.

So, the image contains node, linux alpline, and our application file. Now if we use a different node image that was based on different distribution of linux. We would endup with the large image and when deploying the image we would have to transfer that image from one computer to the another. That's why we use linux-alpline which is very small image.

And then in the the image uploaded virtually, we just need to type command docker run first_docke

Distros

- Ubuntu
- Debris
- Alpine
- Fedora
- centOS

docker ps (shows list of process means containers running)
docker ps -a (show all container like stopped, and running containers)

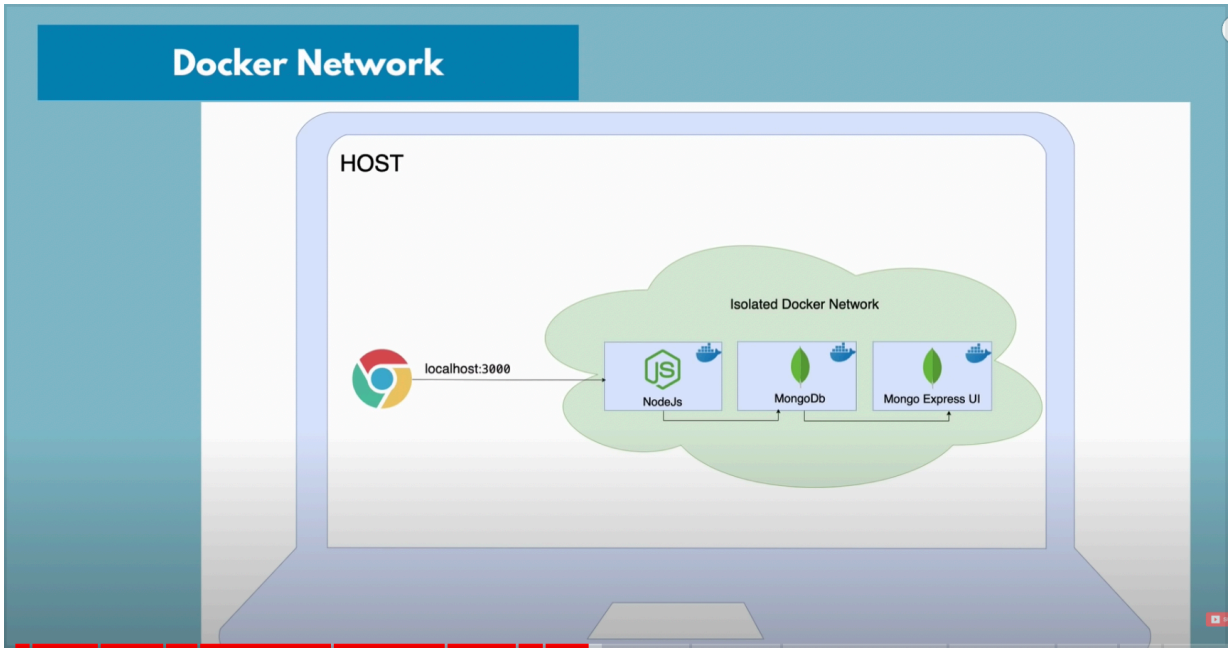
Package managers

- Nom
- yarn
- Pip, and so on..

Binding the docker container port to the host port
Docker run -p6000:6379 redis

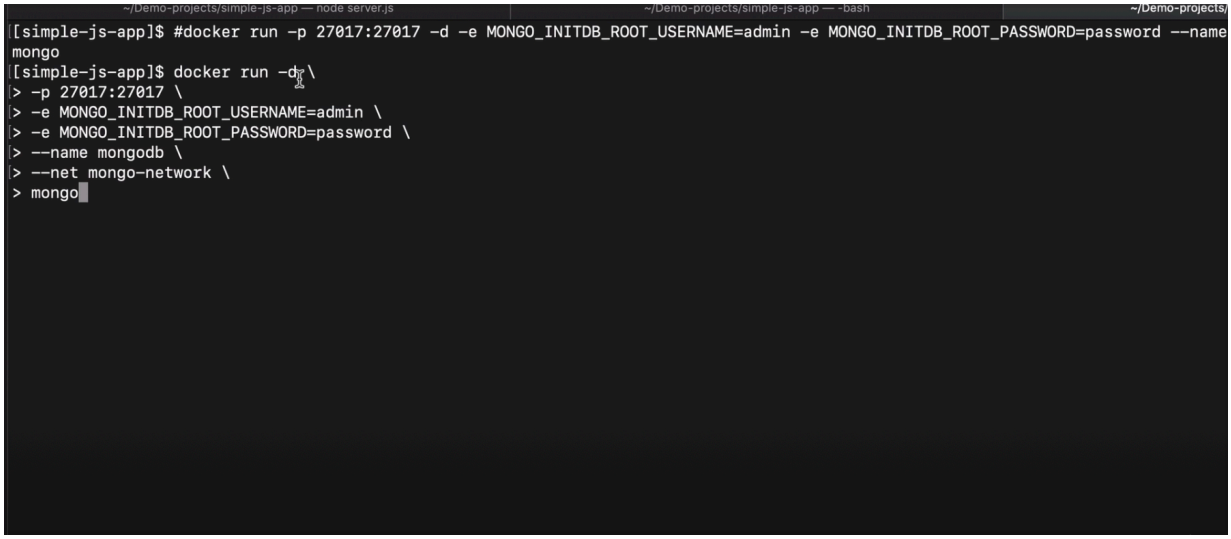
Go inside of the container command
- Docker exec -it container-id /bin/bash
- Docker exec -it container-id /bin/sh

Docker Network

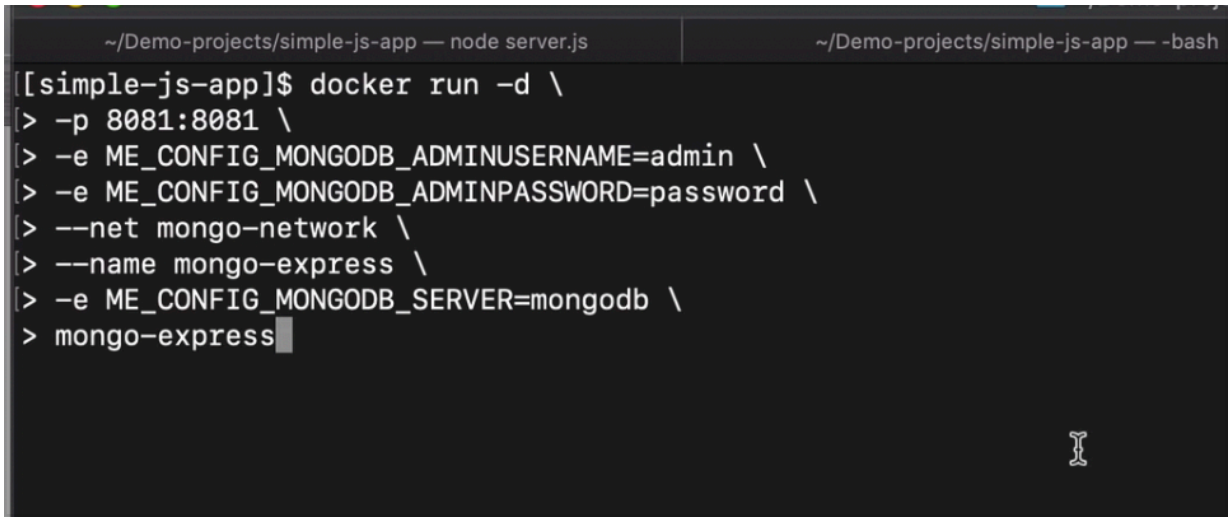


Command to create network: docker network create mongo-network

- docker run -p 27017:27017 -d -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin --name mongodb --net mongo-network mongo



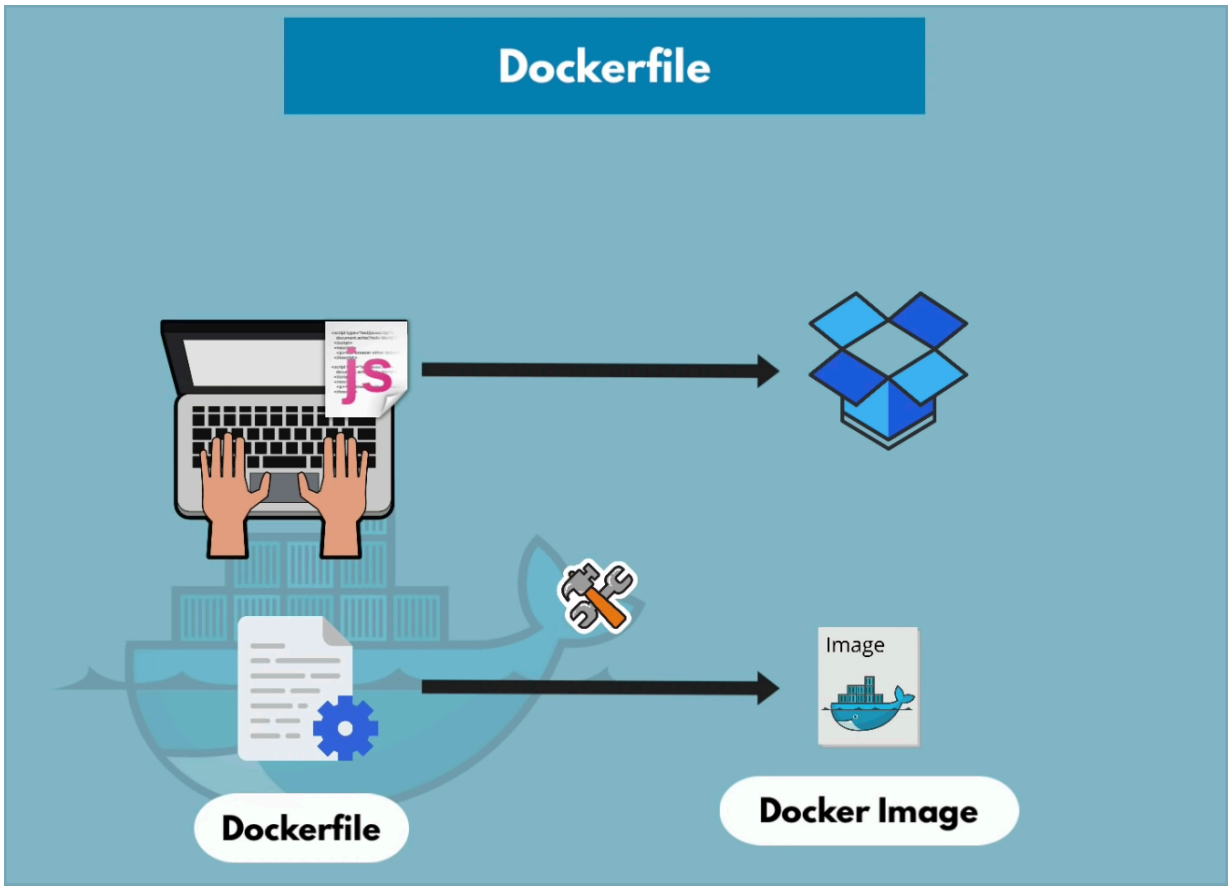
- docker run -d -p 8081:8081 -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin -e ME_CONFIG_MONGODB_ADMINPASSWORD=admin --net mongo-network --name mongo-express -e ME_CONFIG_MONGODB_SERVER=mongodb mongo-express

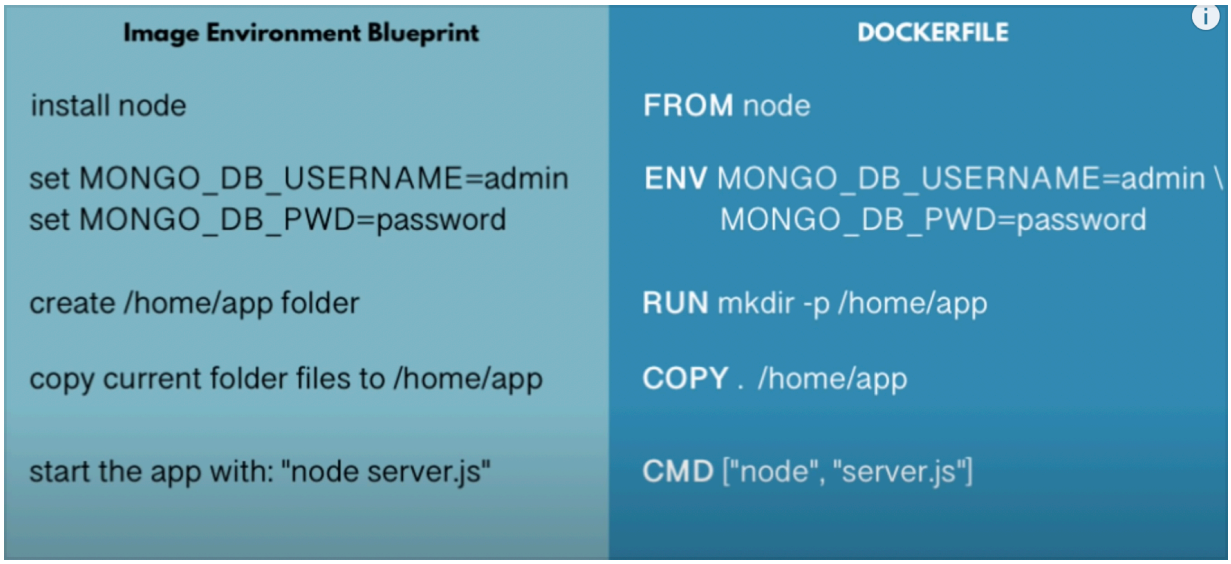


Docker compose:

- docker-compose -f name.yml up (start all the container that mentioned in compose.yaml file)
- docker-compose -f name.yml down (stop all the container that mentioned in compose.yaml file)

Deploy:

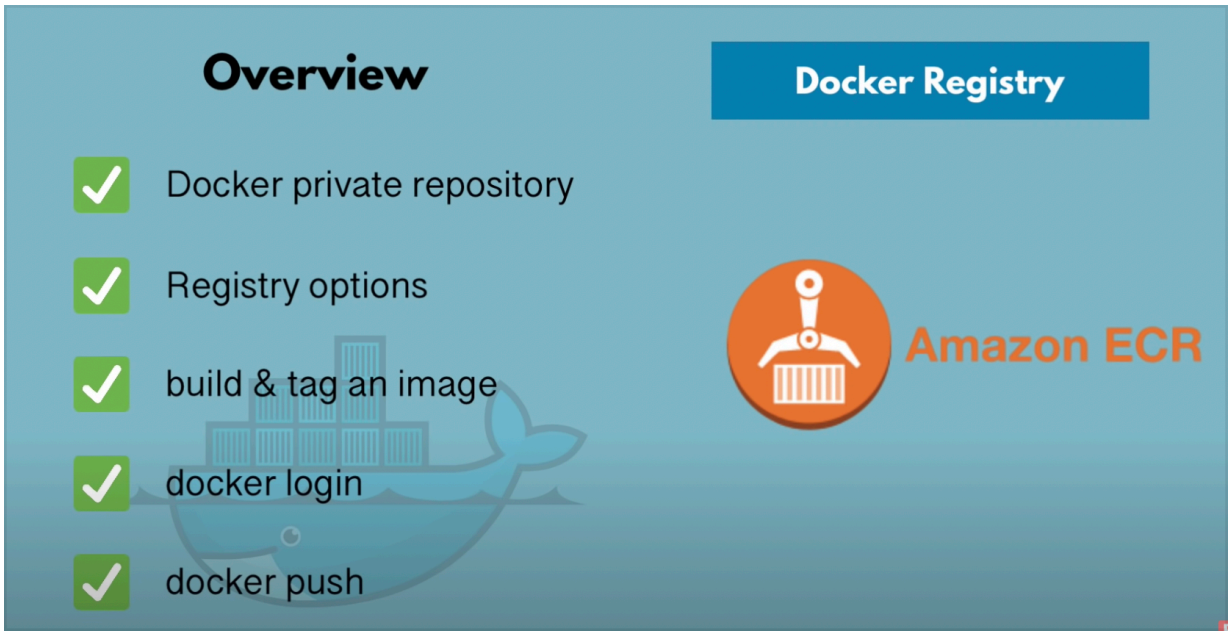




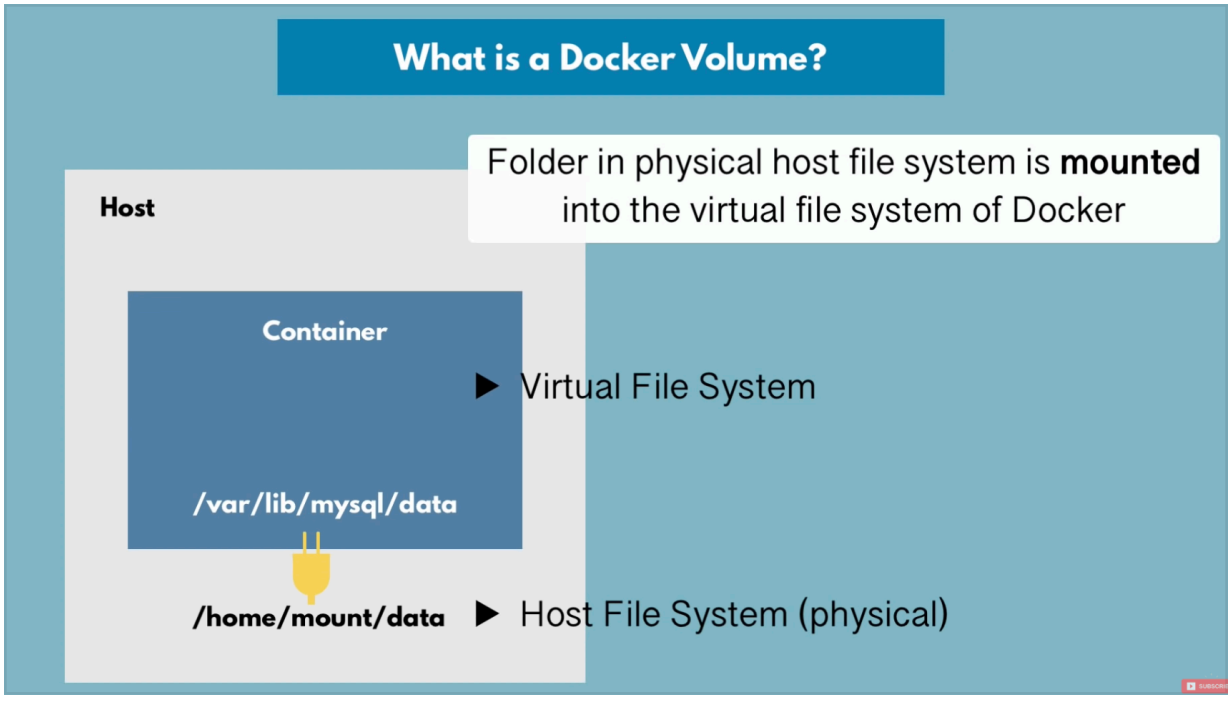
- docker build -t my-app:v1.0 . (. defines the source)
- docker run my-app:v1.0

Delete the docker container: docker rm containerID
Delete the docker image: docker rmi imageID

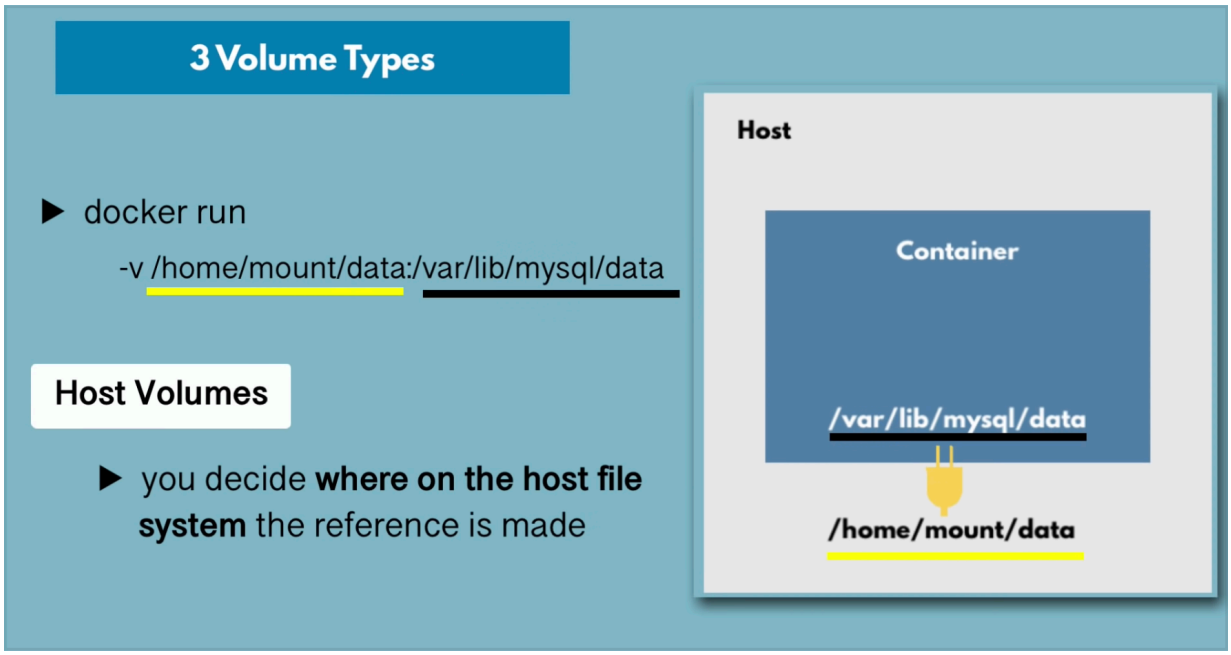
Docker private repository:



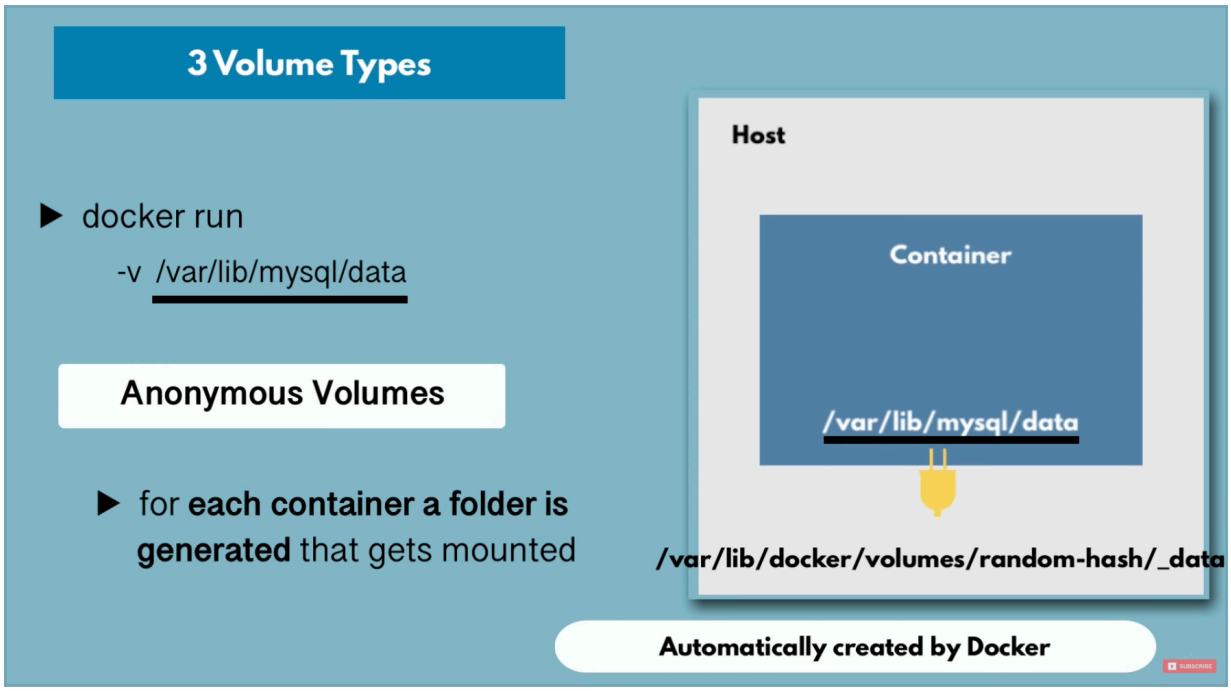
Docker Volume:



1st type



2nd type



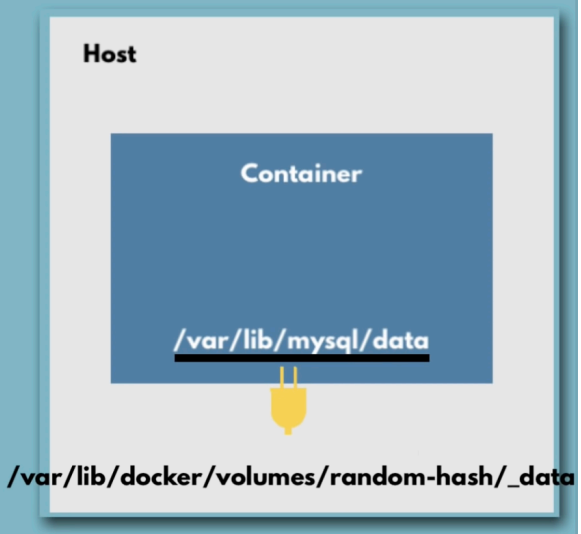
3rd type (mostly used in development)

3 Volume Types

- docker run
  -v **name:**/var/lib/mysql/data

Named Volumes

- you can **reference** the volume **by name**



Docker Volumes in docker-compose

Named Volume

mongo-docker-compose.yaml

```
version: '3'

services:

    mongodb:

        image: mongo

        ports:
         - 27017:27017

        volumes:
         - db-data:/var/lib/mysql/data

    mongo-express:

        image: mongo-express
        ...

volumes:
    db-data
```