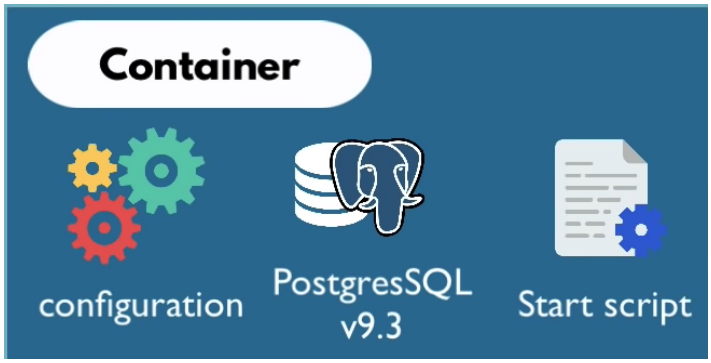
Docker

Container-

- A Docker container is a lightweight and executable package of software.
- It includes everything used to run an application, code, libraries, and dependencies.
- You don't need to allocate any memory for the application. It can automatically generate space according to the requirements.
- Portable artifact, easily shared and moved around
- Layers of images

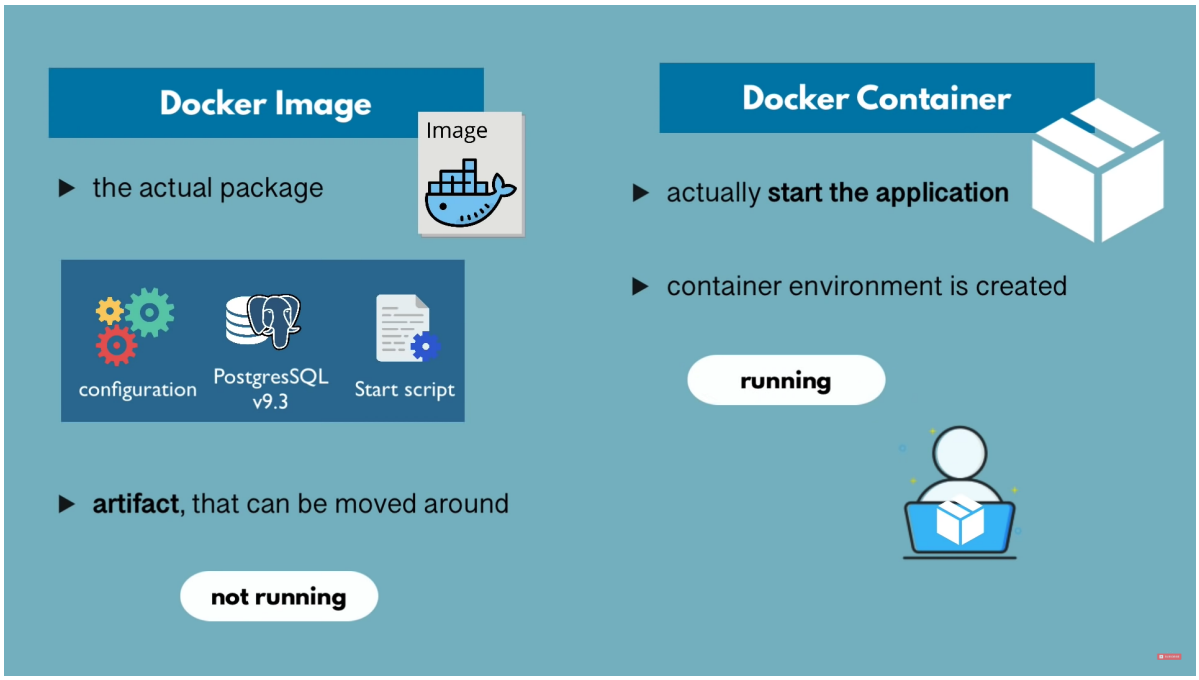
Where do containers live??

Container lives in container repository either in a *Private Repository* or in a *Public Repository (DockerHub)*



NOTE :- You can run one application existing with two different versions in the system without any conflict using docker. Eg:- I can run postgres version 9.6 and postgres version 8.6 both simultaneously in the system.

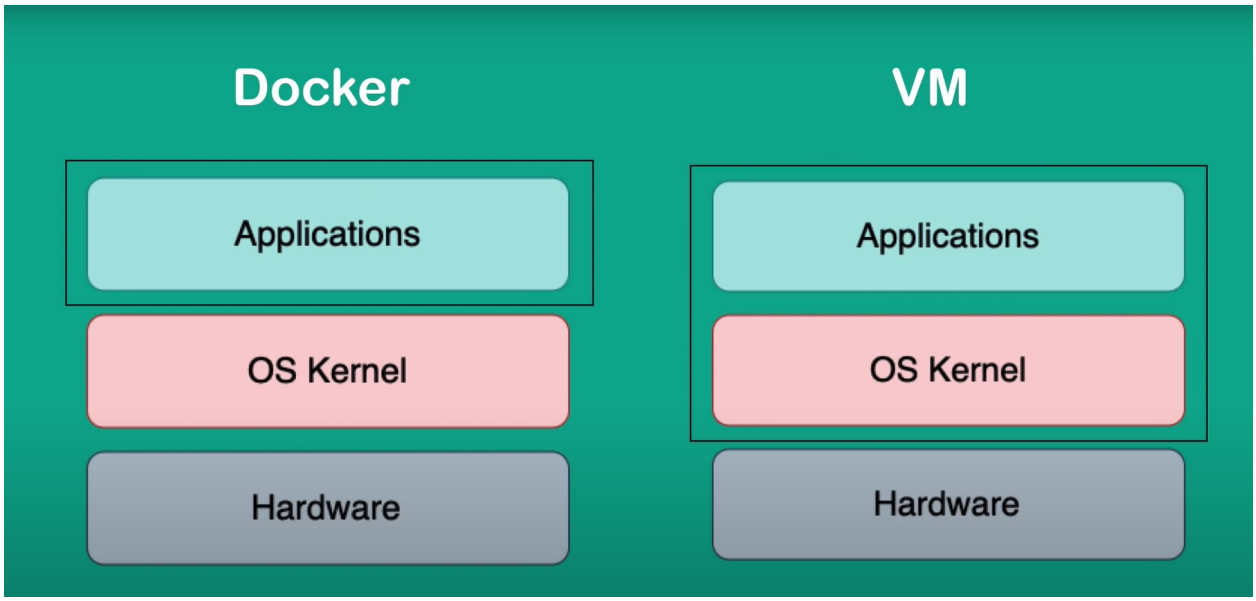
Docker Image vs Docker Container



Docker VS Virtual Machine

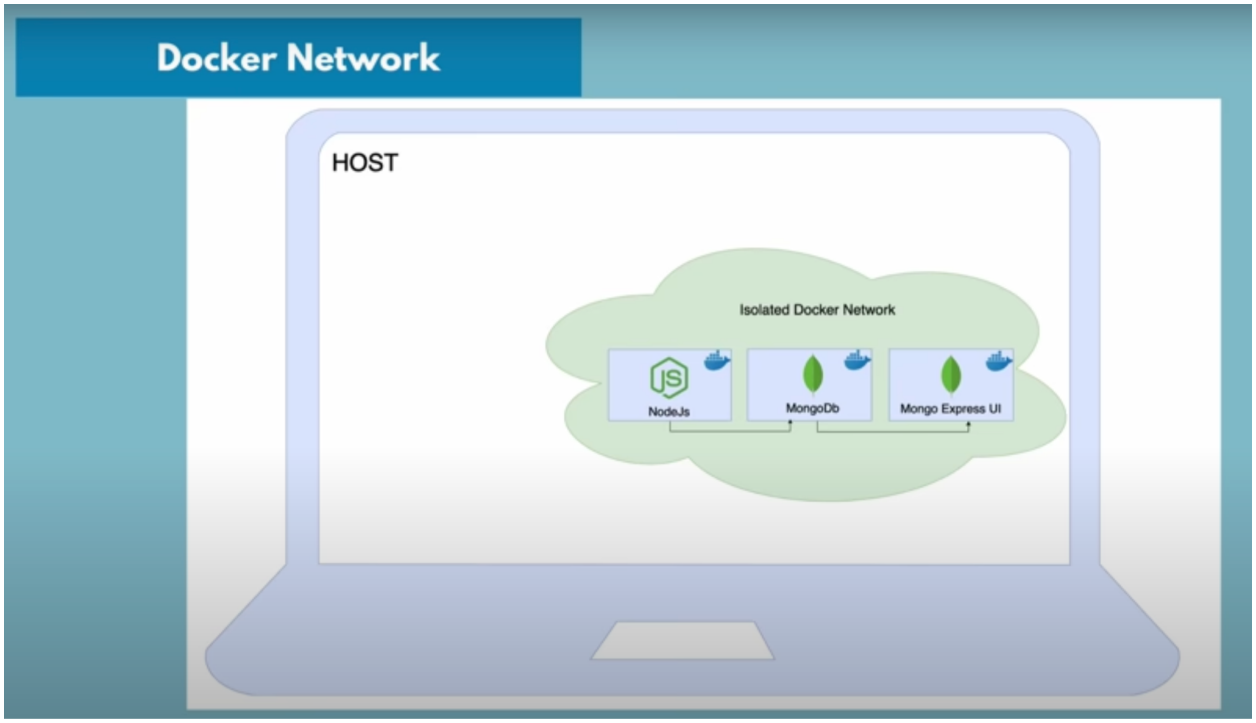
Docker virtualizes the application's layer and it uses the kernel of the host because it doesn't have its own host

Virtual Machines on the other hand have their own kernel and applications layer.



Docker Network

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



Basic Docker Commands:-

docker run postgres:9.6

- It looks if the postgres with version 9.6 exists locally on the system or not, if not then it pulls the image of the container from the docker-hub and then runs it.
- If you don't specify the version then it automatically installs the latest version.

docker ps
It lists down all the running containers in the system.

docker run -d redis
Runs the container in the detached mode

docker stop <container_id>
Stop the running docker container. Here you can obtain the container_id through **docker ps** command.

docker start <container_id>
Starts the running docker container. Here you can obtain the container_id through **docker ps** command.

docker ps -a
Lists all the running or stopped containers.

docker run -p<host_port>:<container_port>
Eg - docker run -p6000:6379
Binds the container port to the host port, we cannot bind a two container ports to same host_port

docker logs <container_id>
Eg- docker logs d7a678990rf9
It gives the logs for the container using container_port

docker logs <container_name>
Eg- docker logs dreamy_ball
It gives the logs for the container using container_name

docker run -d p6001:6379 - -name redis-older redis:4.0
If we want to change the name of the existing redis version 4.0 container then we can run the above command and rename the redis:4.0 to redis-older.

docker exec -it <container_id> /bin/bash or
docker exec -it <container_id> /bin/sh
This command would allow me to enter into the container's terminal as a root user.

Diff between run and start
Run creates and then runs a new container whereas start is used to restart an existing container.

docker network ls
Lists down all the existing docker networks.

docker network create <network_name>
Creates a docker network

docker logs <container_id> -f
This will help to stream the logs by marking the log using "—-" to mark the beginning of the logs from the end.

docker logs <container_id> | tail
It displays the last log or end part of logs

docker system prune
Removes all stopped containers

docker rm <container_id>
Removes a specific container

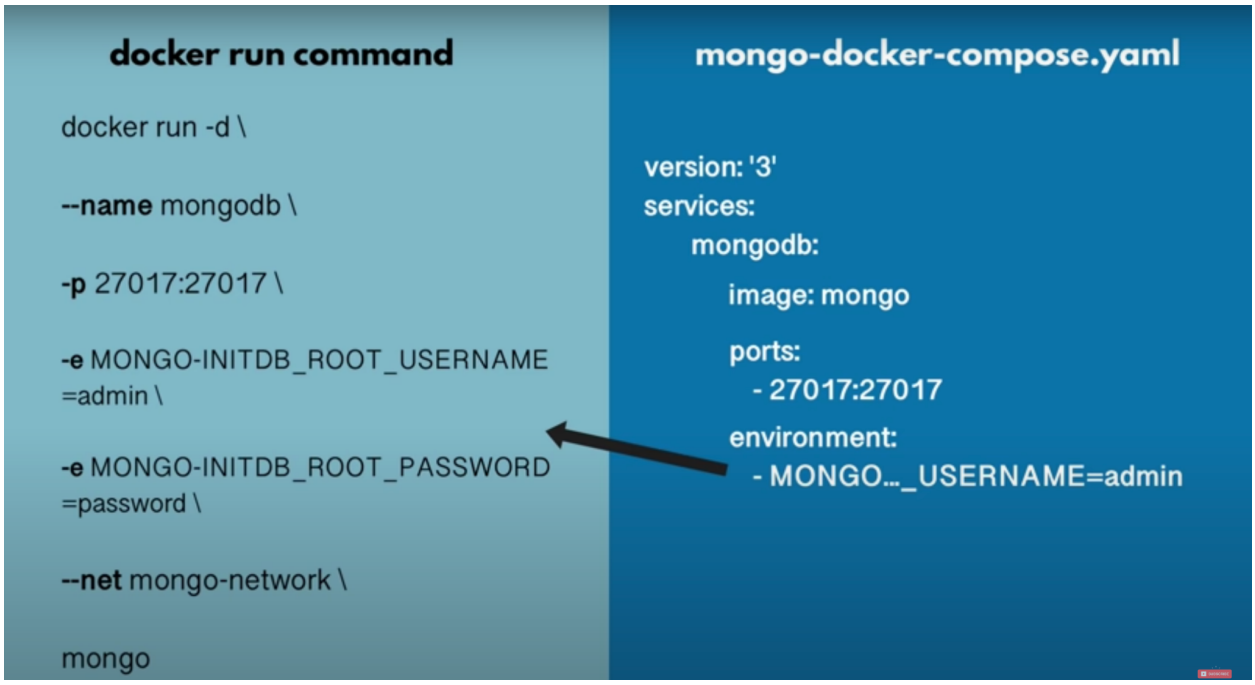
docker rm -f <container_id>
Forcefully stops a running container

docker cp <source> <destination>
With the above command one can copy any file from suppose container(source) to their own system(destination)
Eg- docker cp 1f12:harry.txt vanshika.txt

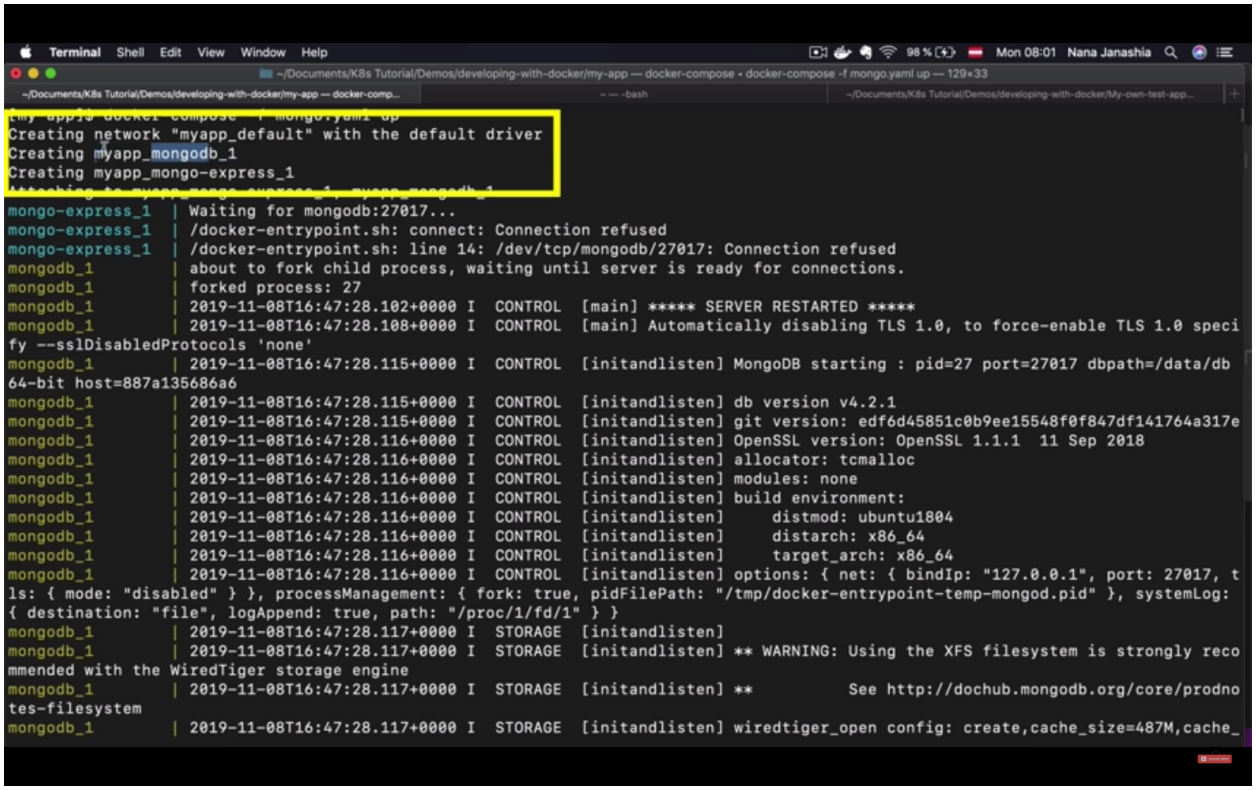
docker commit <container_id> <image_name>
Suppose you've modified a container and now you want to create an image for it then you can commit the changes using the above command and it'll create an image.

**docker run -d ** —> running the docker container in detached mode
**-p 27017:27017 ** —-> mongodb port : 27017
-e MONGO_INITDB_ROOT_USERNAME=admin \ → overwriting usrname
**-e MONGO_INITDB_ROOT_PASSWORD=password ** —> overwriting pass
**- -name mongodb ** —> overwriting container name
**- -net mongo-network ** —-> overwriting network name
mongo → mongodb container name
-e → to set environment variables

Docker Compose
Suppose we have multiple docker commands like above to run so writing the whole command like it for each container is a bit tedious so to write this in a structured way we use docker compose.



Docker compose takes care of creating a common network.



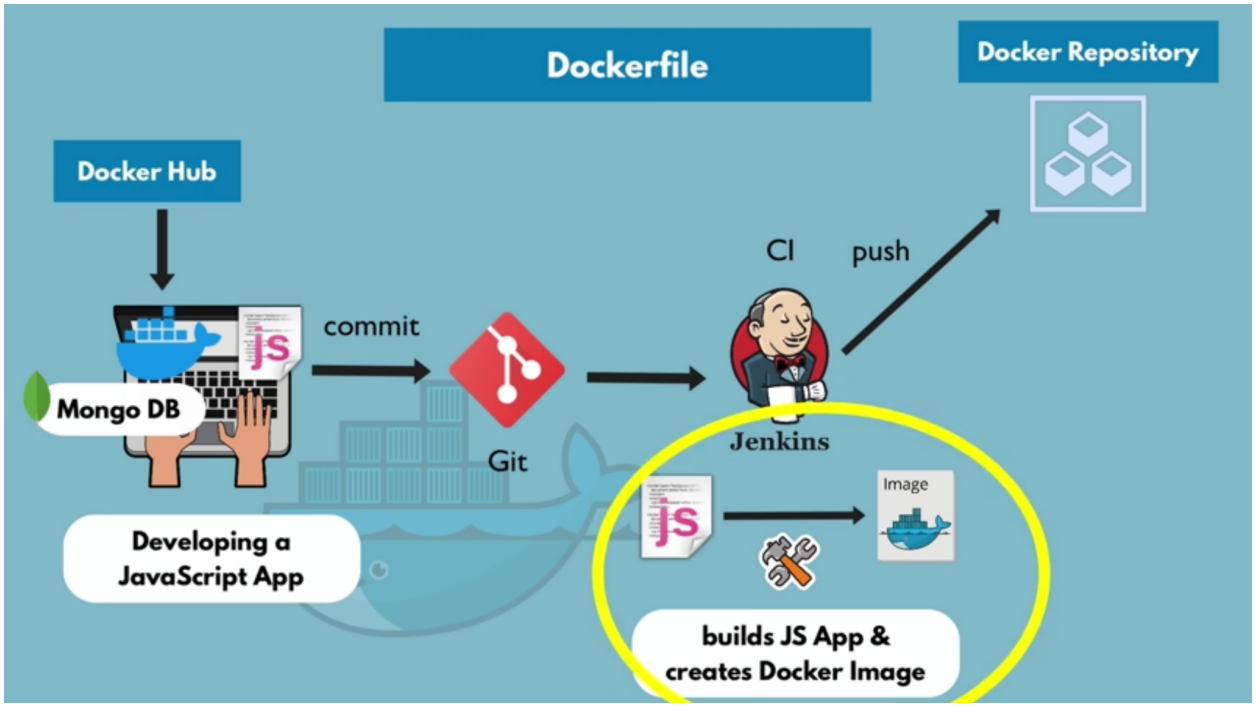
docker compose -f mongo.yaml up

This command will start running all the containers defined in the mongo.yaml file.

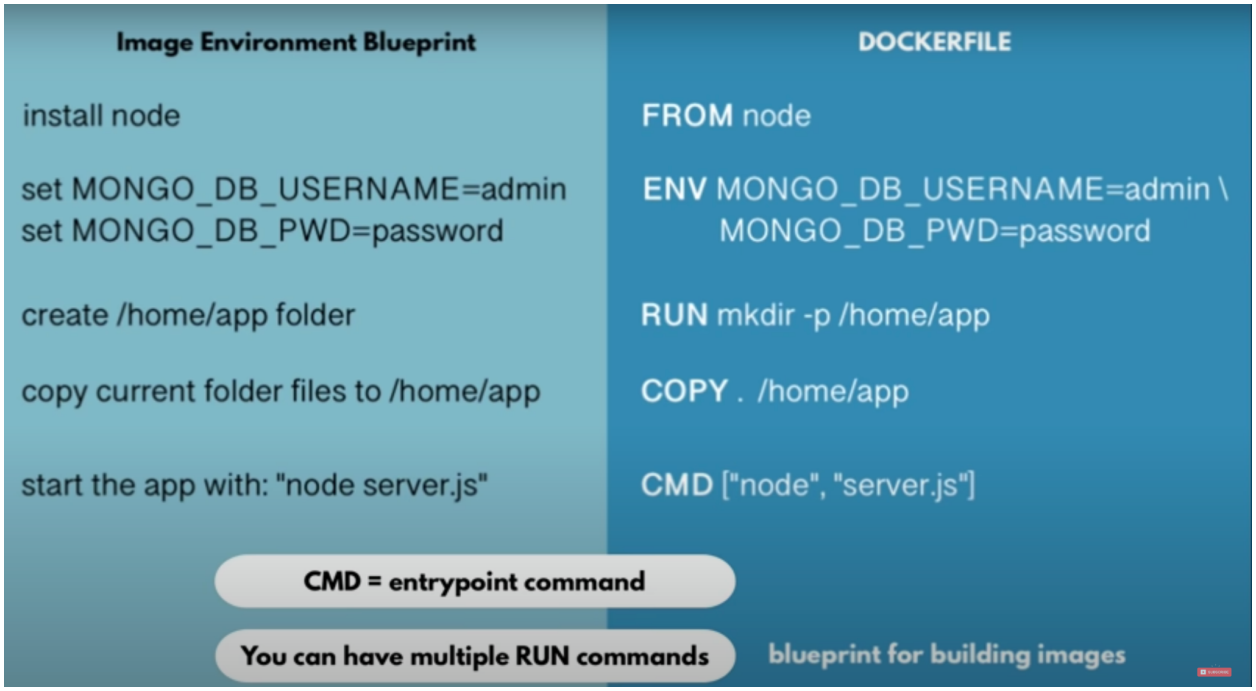
docker compose -f mongo.yaml down

This command will stop running all the containers defined in the mongo.yaml file. Also it removes the default network created at the time of *docker compose -f mongo.yaml up* command.

Dockerfile : Building our own docker image

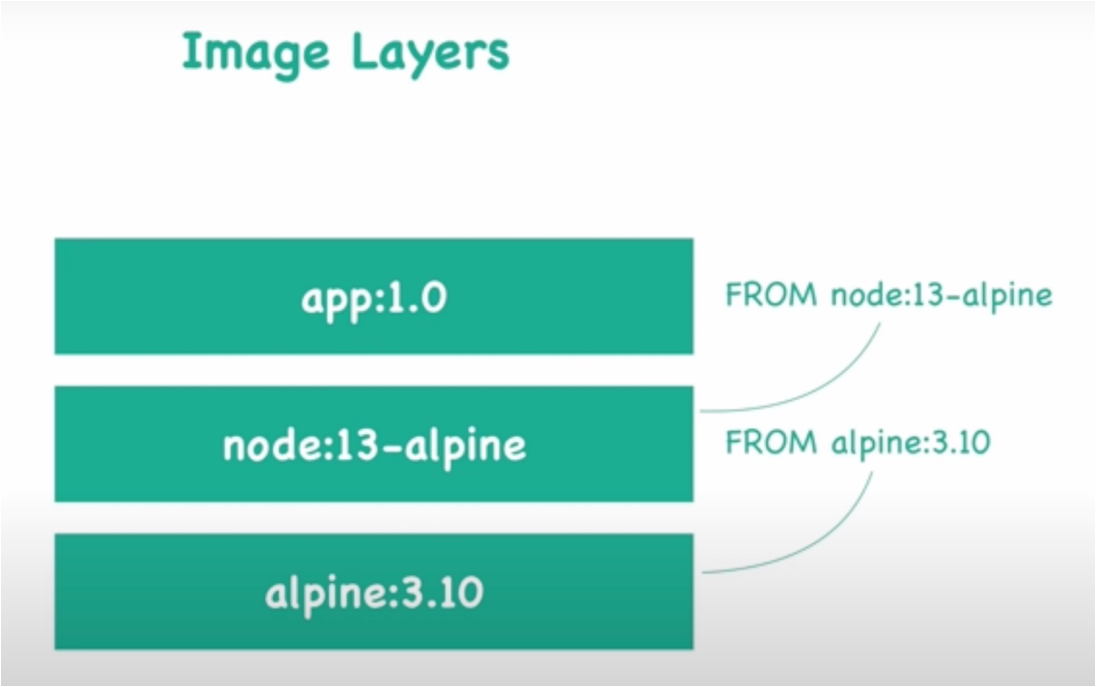


Docker file is a blueprint for building docker images.



NOTE ➖

- FROM node - base image
  Node is pre-installed because of the base image.
  We can specify the node version as FROM node:13-alpine
- RUN - execute any linux command
  Directory is created inside of the container
- COPY - executes on the Host machine
  COPY <source> <destination>
  For executing inside the container we would've used run cp <source> <destination>
- CMD ["node","server.js"] = node server.js
  It's an entry point command used once, while RUN can be used multiple times.

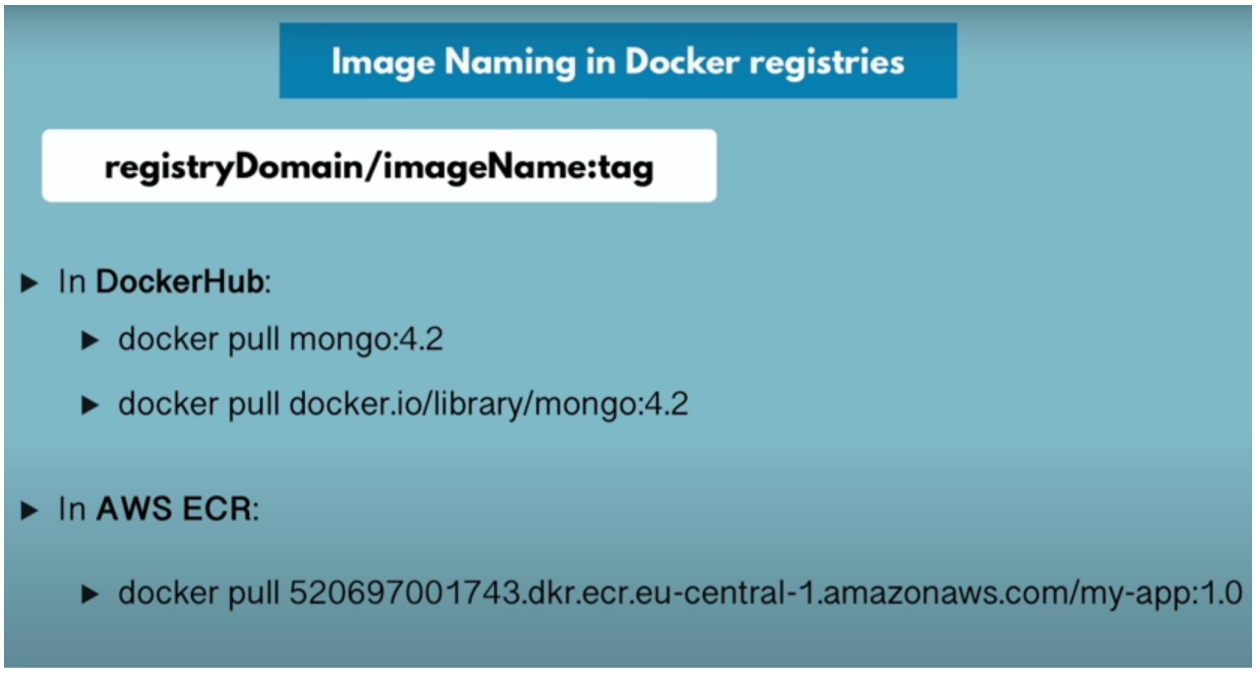


docker build -t <image_name>:<tag> <location_of_the_dockerfile>
This command would create a docker image provided with the location of docker-file.
Eg- docker build -t my-app:1.0 .
Here "." signifies current directory

IMP - Whenever you modify the dockerfile you must rebuild the image.

docker rmi <image_id>
First we should remove the docker container using docker rm <container_id> and then we can remove the image of the container using the above mentioned command.

Pushing our built Docker Image into a private Registry on AWS



- ❖ Since till now we were building the image on docker-hub we didn't require to provide registryDomain but since now we're pushing our image to private aws repository so we'll need to provide the registryDomain and shown in image above.
- ❖ In Amazon ECS, there is one repo per image and each repo contains different versions and tags of the same image.
- ❖ Before pushing the image into a private repo it requires docker login for authentication and the prerequisites for that is AWS CLI must be installed and credentials configured.
- ❖ The push commands can be seen in AWS ECR

- Since we cannot directly push by using *docker push my-image:1.0* as it would push the image to docker hub and not the repo so for that we tag i.e rename the image.
- Docker tag basically creates a copy of the image with the renamed name as follows :-

docker tag <docker_image_name> <docker_rename_name>:<version>