Docker

Container is a running instance of a image and from one image you can run multiple containers

**SO basically you build your code**

**create image from docker FIle with your code inside**

**RUN application with docker compose file with all depended images and your app image inside**

# Basic Docker CLIs

| **Command** | **Description** |
| --- | --- |
| **docker run** | **Run a command in a new container** |
| **docker build** | **Build an image from a Dockerfile** |
| **docker push** | **Push an image to a registry** |
| **docker pull** | **Pull an image from a registry** |
| **docker stop** | **Stop a running container** |
| **docker rm** | **Remove one or more containers** |

# Commands

**docker images** will list all images

**docker ps** will list all running containers **-a** is added will show stopped container also

Docker Registry:- Ready docker images available in image storage “Docker Hub”

**docker pull <**imageName>:<ImageTag> to pull image from docker registry

docker pull nginx:1.23

**docker run <imageName>:<tag>** will run docker image in container **--name** to provide name **-d** for detached

**cntrl + C** container will exit/stops and if add -d on above run it will run

container in detached mode

**docker logs** <container ID> will show logs of container -f to stream logs | tail for last lines

**port binding is need when a web app is running in container so its not accessible in**

**outer network to avail that we need port binding ,because app in running inside container**

**port not on local host port so we need to bind both ports**

**eg:-** docker run -d **-p 9000:80** nginx:1.23 **local host port 9000 is bind with container port 80**

**docker stop** <Container ID>

**docker start** <Container ID> run already created container.

**docker rmi <image id>** to remove image

docker run command always create new container everytime does not use older containers

# Private Registries in docker

Aws has ECR registry for private container images Elestic container registry in ECR we can have multiple

Versions of same image stored in repository

**docker exec** if some thing wrong with container and I want to open a terminal inside it to check

or I want to check config file in it eg **docker exec -it <containerID> /bin/bash**

it is iteractive terminal type **exit** to come out

# Build Image :

To create custom image you need

**Docker File** is a file that contains commands to assemble image, docker can then build a image from

These instructions.

We need to provide definition to create image in docker file

We need a base image to build our docker image from mostly a light weight linux os image

Then say java installed in it

We define base image with **FROM** keyword in docker file

1)FROM node:19-alpine (base image:tag *linux image with node in it* )

Now we need to install our app dependency in it so RUN is used to do that

Run is used to execute any command in shell inside container.

3) RUN npm install

Before RUN we need our app code inside image also and RUN will read our json depndecy from

In so line 2 will be

2) COPY package.json /app/

COPY src /app/

WORKDIR /app This will change current dir to be app now we can run

RUN npm install

CMD [“node”,”server.js”] instruction to be executed when container starts only one per dockerFile

Now we have thi complete docker file and in order to build image out of it

We need to run below docker command

**docker build -t node-app:1.0 <location of docker file . for current dir>**  -t is for naming image tag

(you can go to the docker file location and use . it will be easy)

Delete an Image **docker rmi <image ID>**

Delete an container **docker rm <container Id>**

Remove all unused images **docker image prune**

*The Docker run “-it” flag is a combination of two options, “-i” and “-t”. The “-i” option runs the Docker image in interactive mode (keep Standard Input Stream) open. However, the “-t” option is utilized to allocate the “pseudo-TTY” terminal*

*to the container*

# Docker Network*.*

Docker creates a network where containers are running in and they can communicate

With just the container name

**Docker network ls** will show all network lists

To create our own network

**Docker network create mongo-network**

**If we** want to use this network with our container let say mongo and mongo-express

We need to provide this while running this above to container as below

**Docker run -p 27017:27017 -d -e MONGO\_INITDB\_ROOT\_USERNAME=admin**

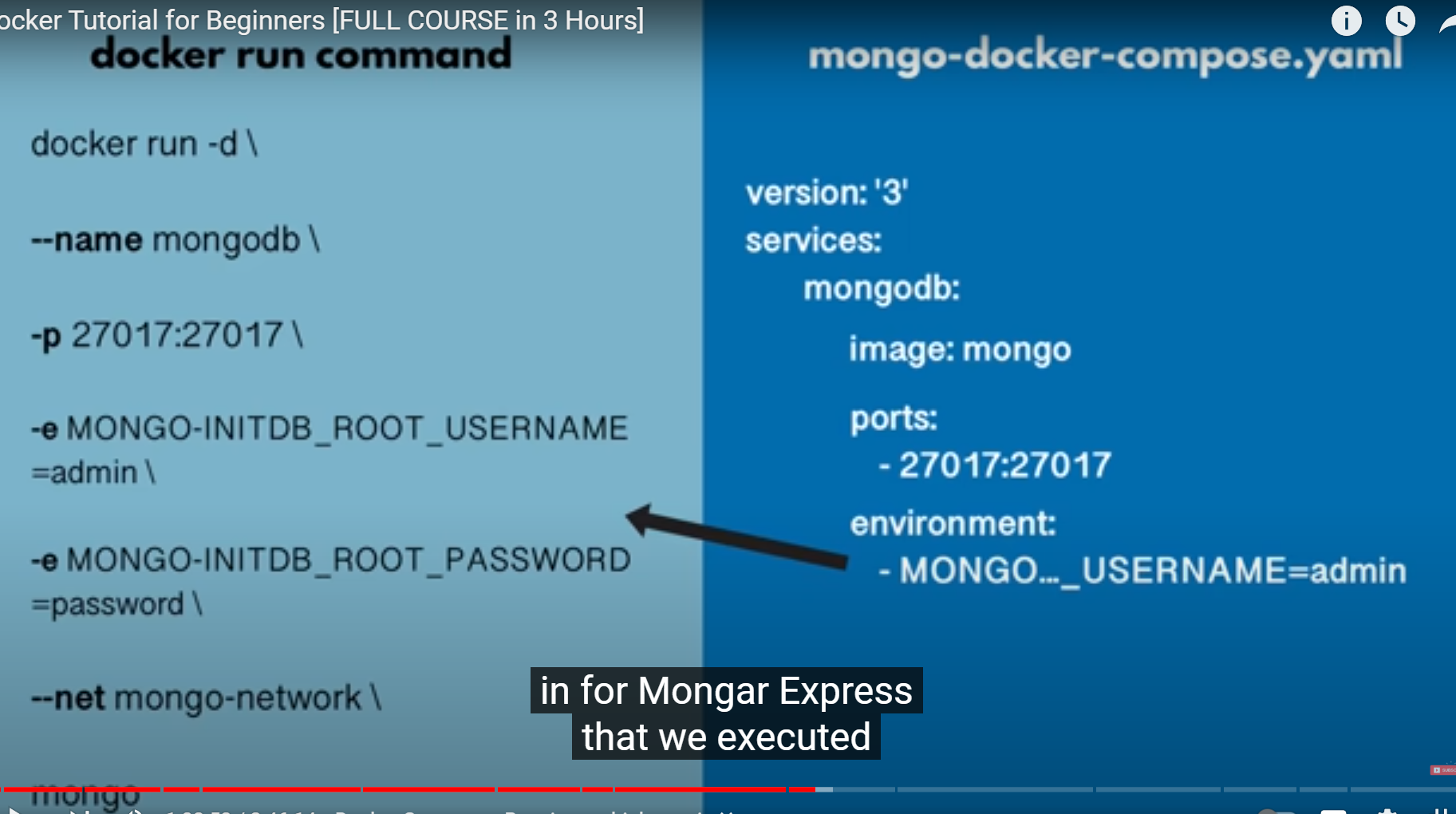
**-e MONGO\_INITDB\_ROOT\_PASSWORD=password –name mongoDB**

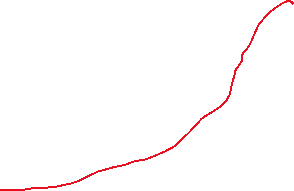
**–network mongo-network mongo**

# Docker Compose

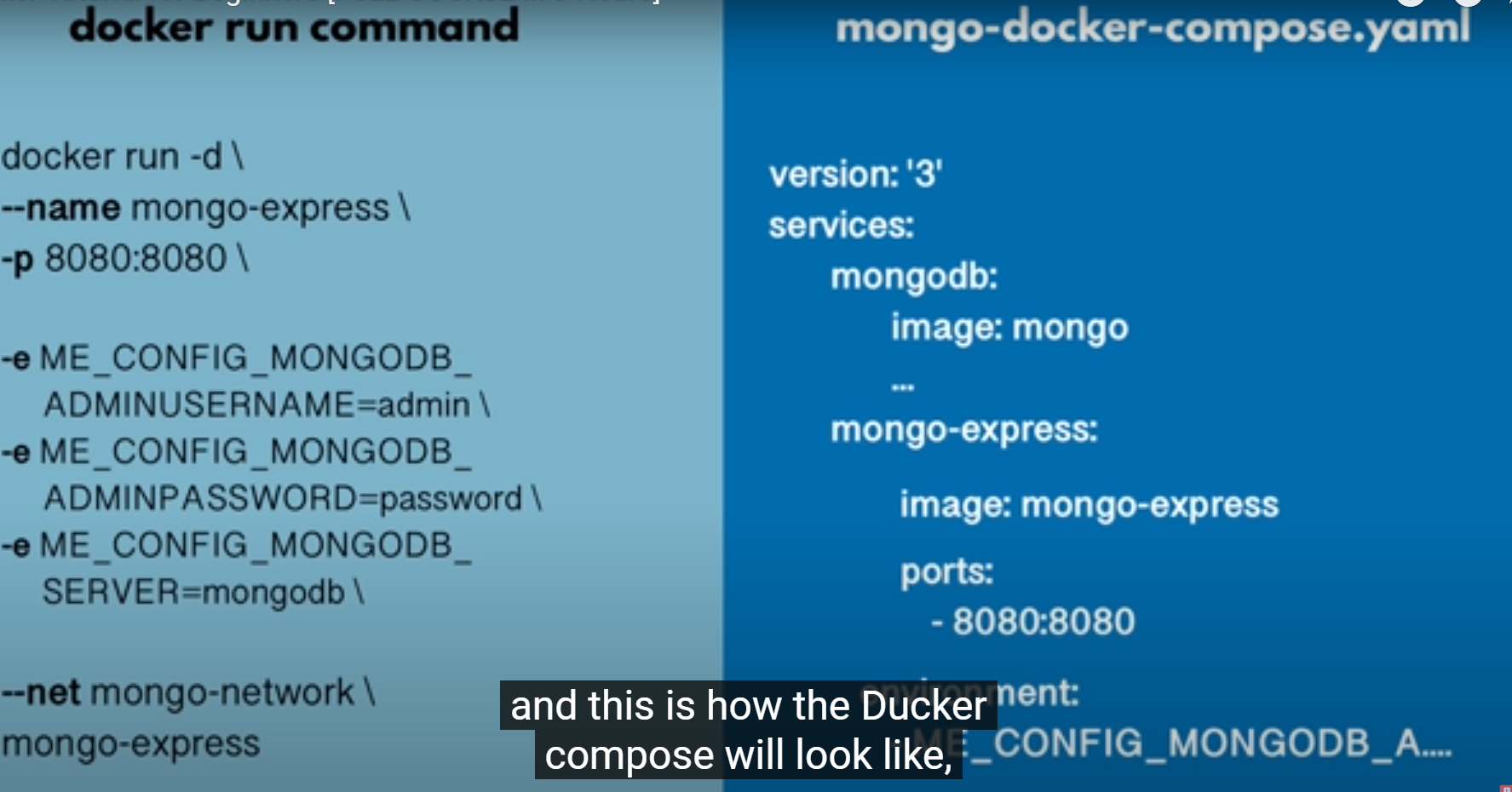
You don’t have to start all containers your self you can create a compose file and that will create

All required **containers with common network** as shown below





mongoDb is container name ,mongo is image ,docker compose will create common network for all containers



Here we don’t have to create network as docker compose will create a common network

To start containers from docker compose file

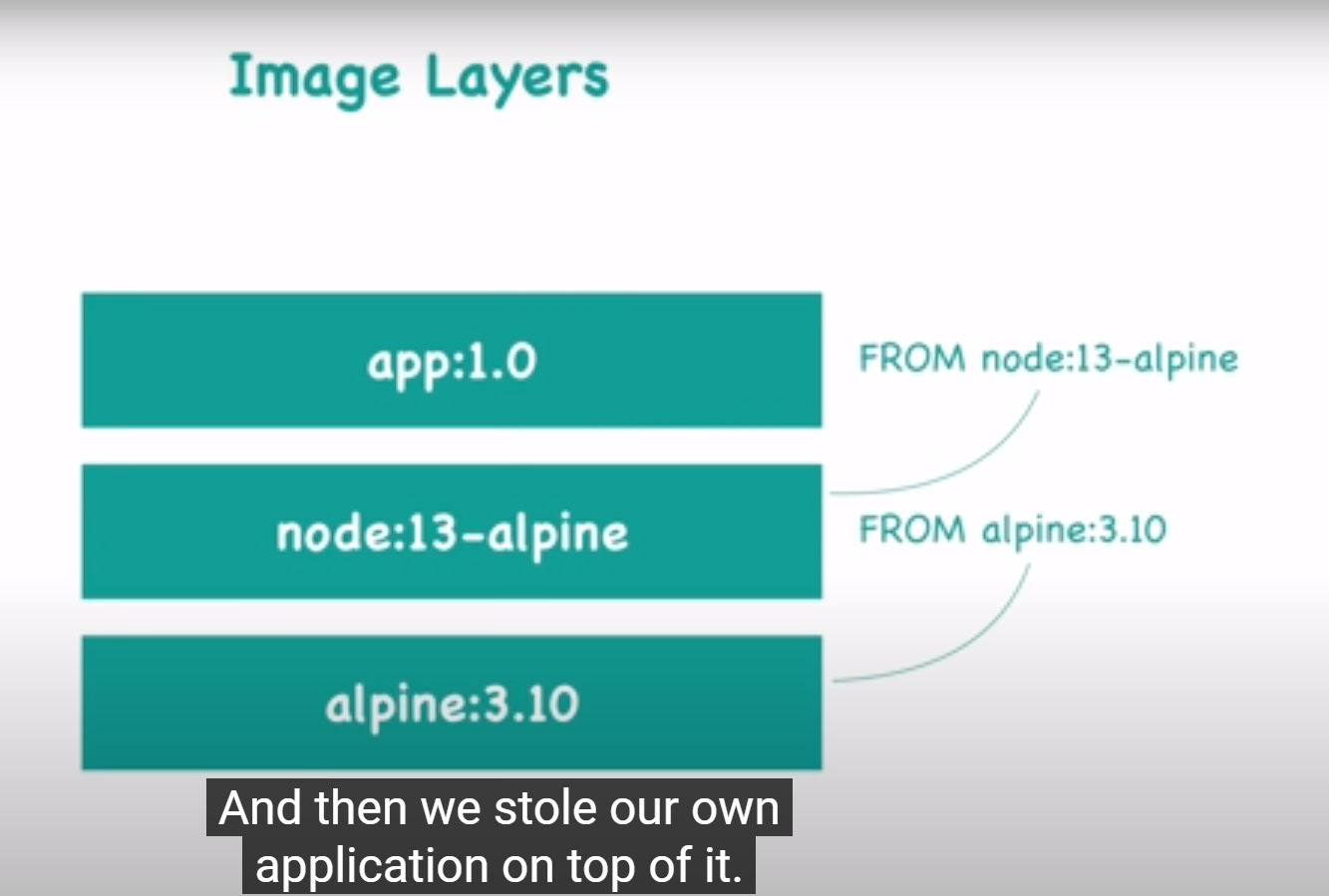
**Docker-compose -f mongo.yaml up** this will start containers

**Docker-compose -f mongo.yaml down** this will stop all containers

# Docker File:-

You Have Developed your application now you need to build a image out of it with our app inside it to distribute

Docker File is a blueprint to build image



These are the layers we will have alpine os image on top of it we have not and then our app

In order to build a image from file we need to provide name and tag and location of docker file



# Login into Container

A screenshot of a computer

Description automatically generated

# ECR Create A Private Repo on AWS for docker images

In Elastic container Registry **(ECR) 2hrs in video**

We need to create a private repository /registry

Here you create a docker repo per image and inside that you can have multiple versions of images

Below are sections of file

FROM:- we need some base to develop our image on in this case node(linux with node installed) from dockerhub

ENV: provide env variable

RUN:- if want to run any linux command inside container

COPY:- if we want copy from our local machine to container then we can use

CMD:-execute entry point linux command

In order to push image to ECR

1. Docker login (command may very for auth)
2. Build image if not build already
3. We need to tag our image with ECR specific name just so that docker will understand that

We are pushing to ECR not docker hub

So tag include that info for image

Docker tag my-app:1.0 <AWS url>/my-app:1.0

1. docker push name\_image:tag

docker file is used to create image and docker compose is used to deploy a image in a server

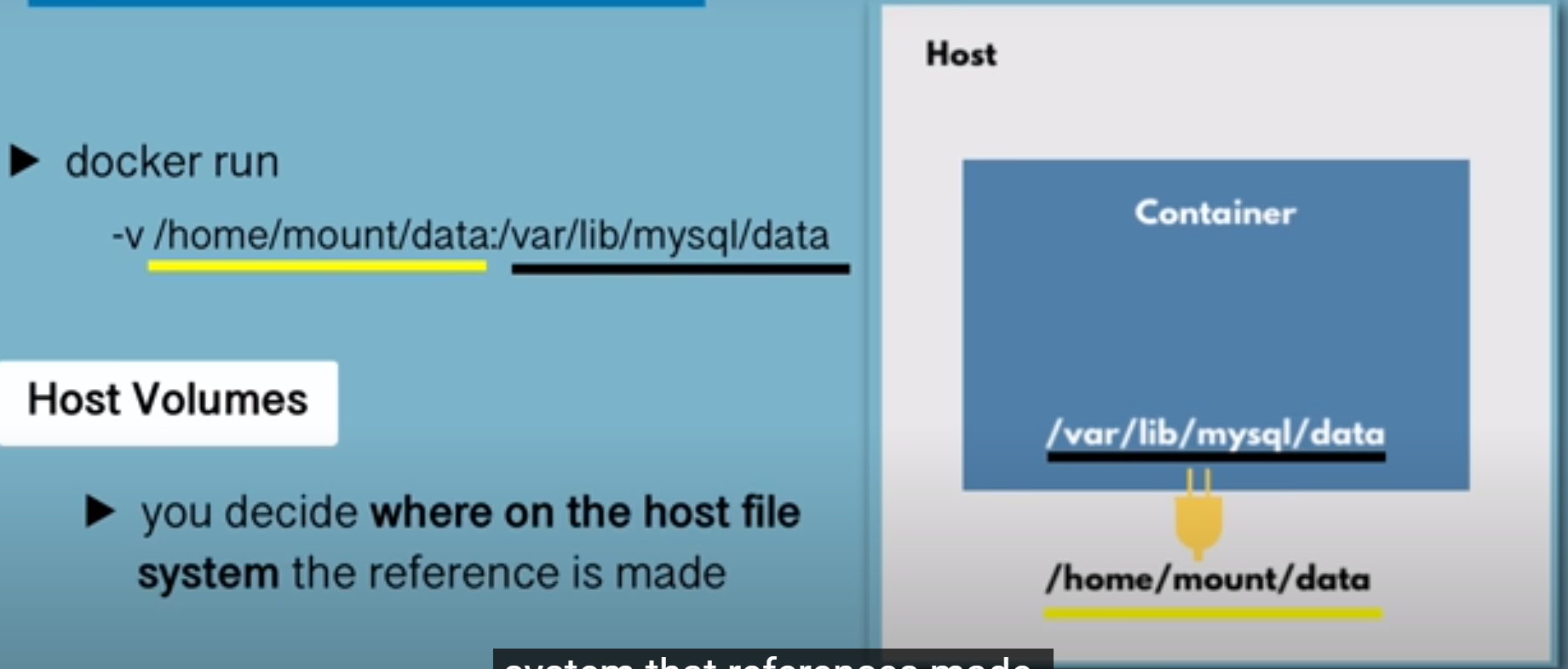
# Docker Volumns

To persist data even if container is killed



3 Types of docker volumns

In docker run command we have -v to define as below 1) HOST VOLUMNS

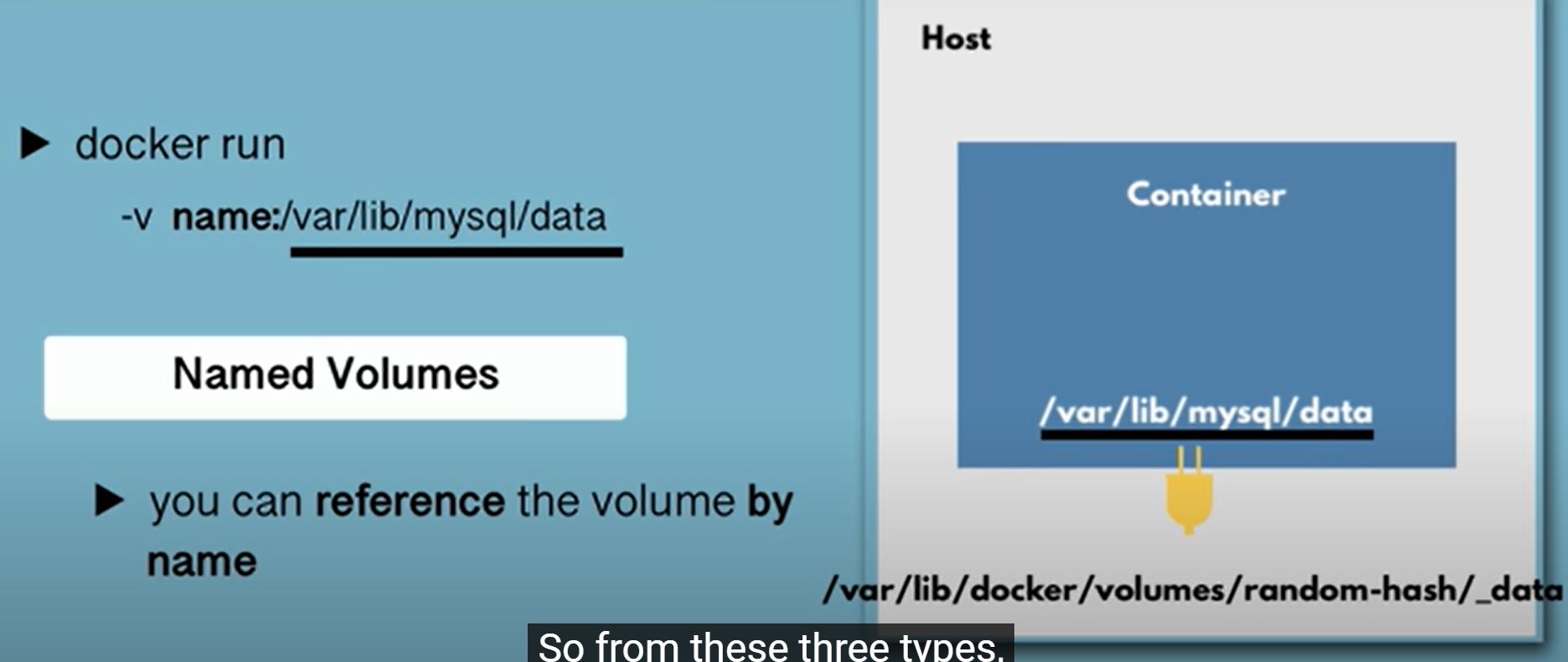


2)Anonymous Volumns :- you don’t provide host dir details

A screenshot of a computer

Description automatically generated

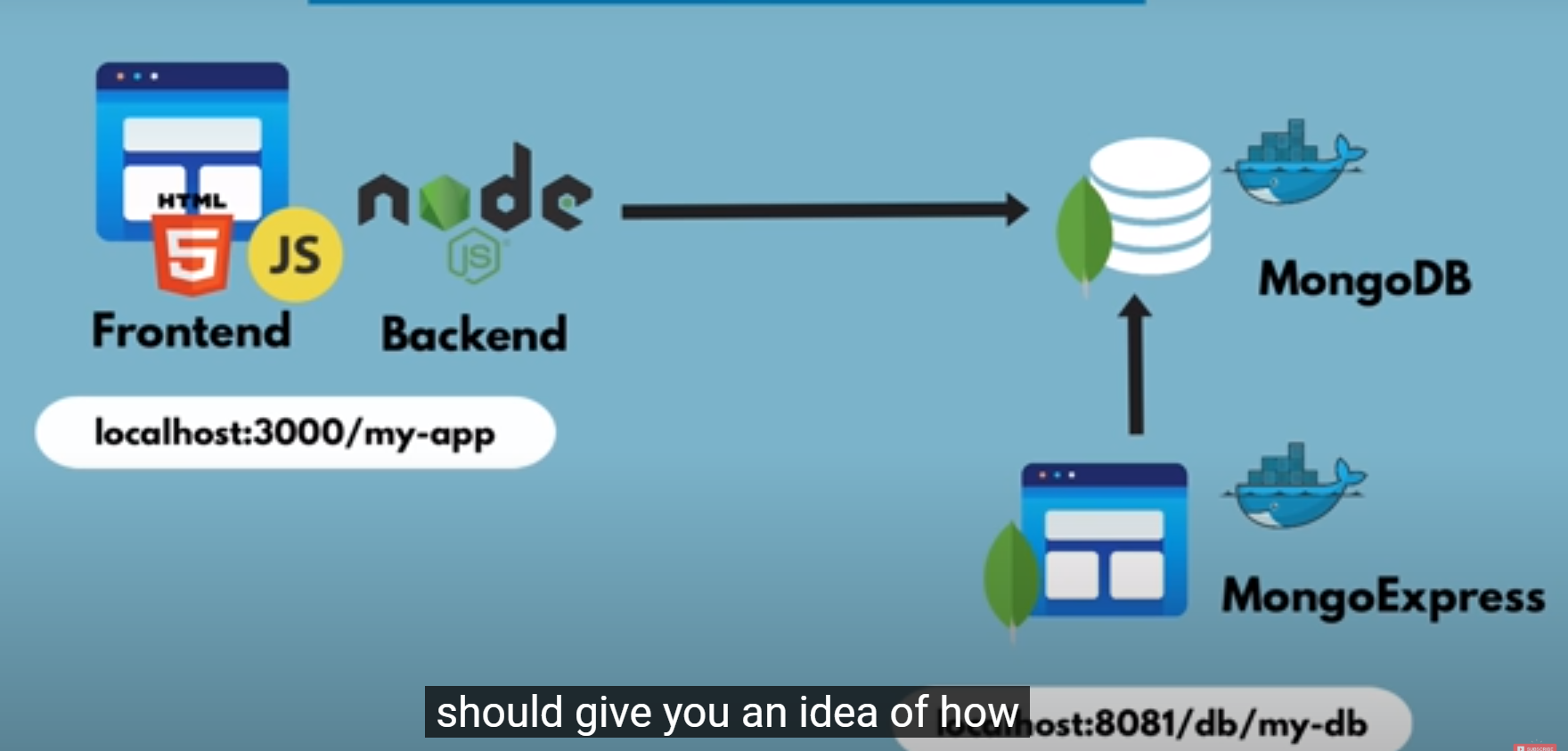
3) Named volumns



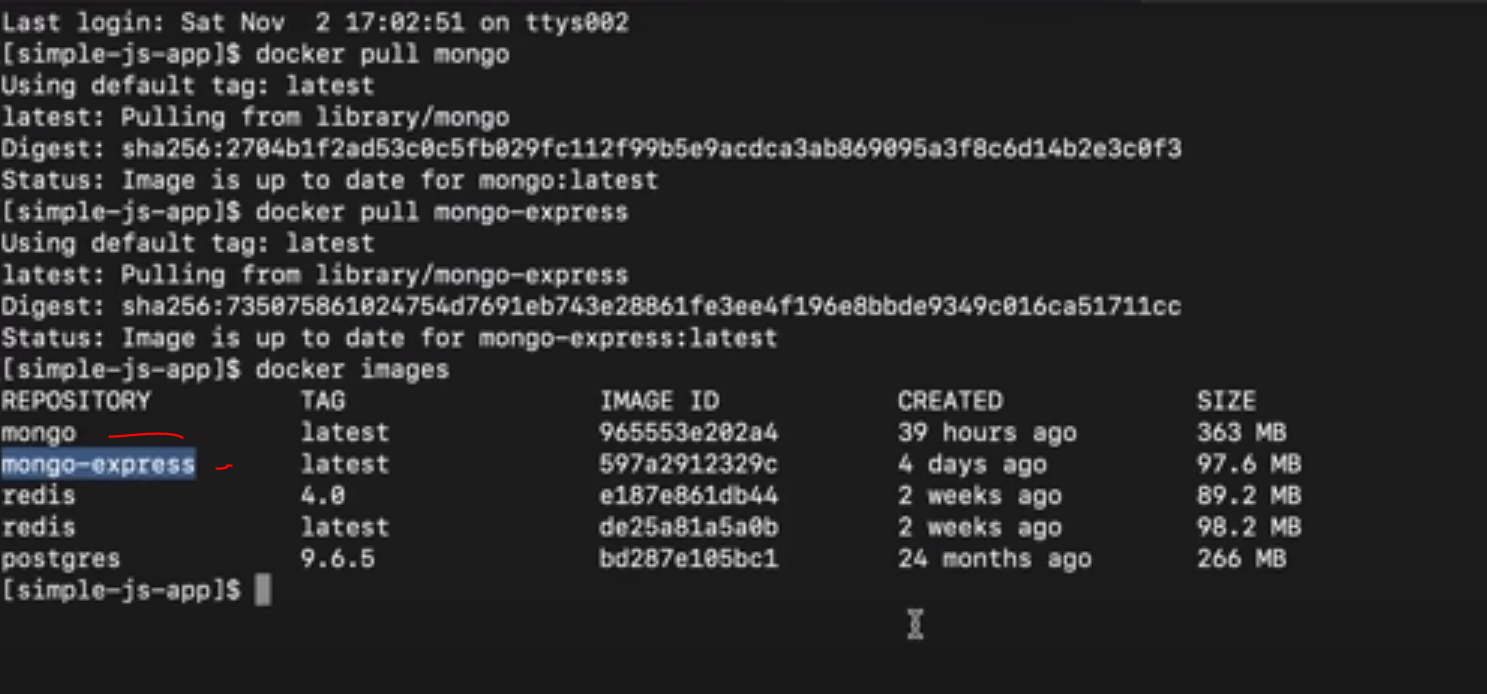
# **EC2 Image Builder**

Used to automate image creation ,maintaining, and securing images

# SMALL APP

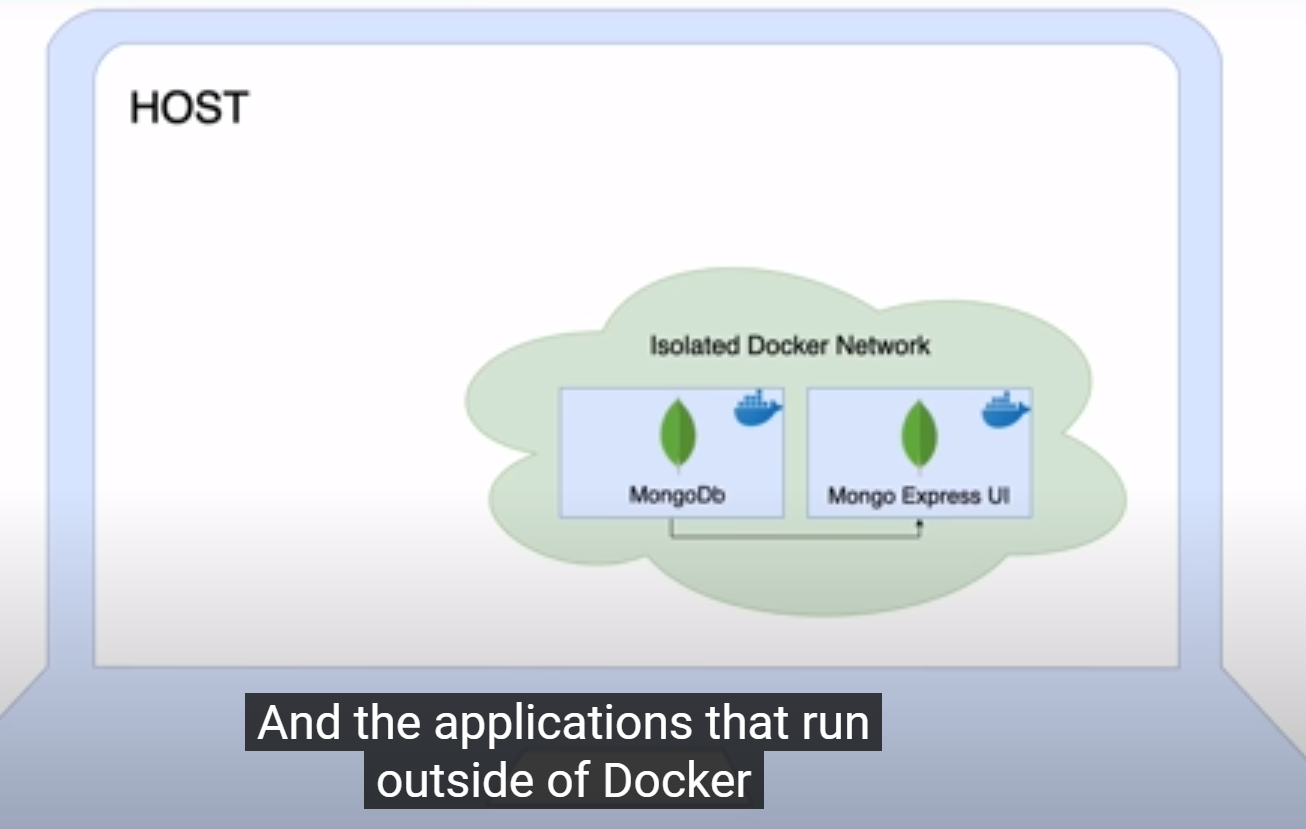


Pulled MongoDB and Express(UI to see DB)

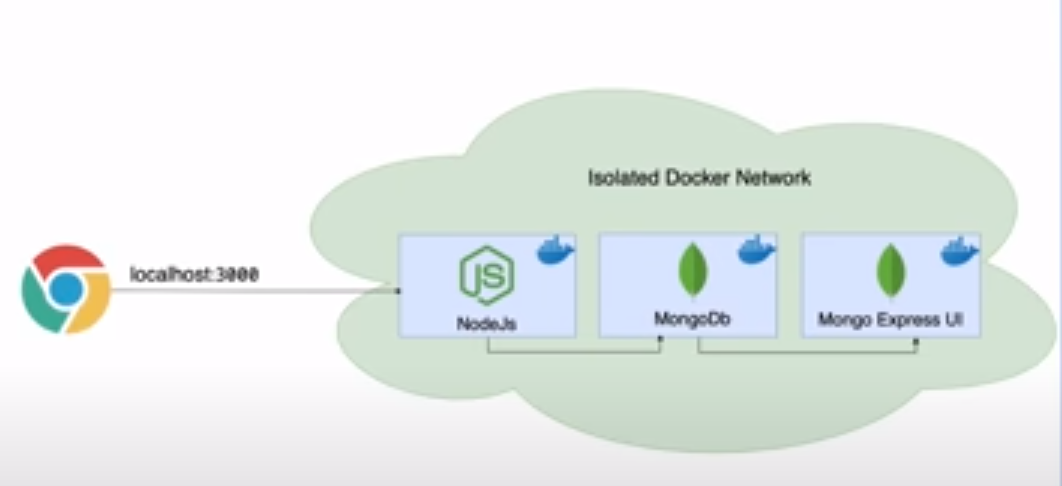


Now we need to connect both so we need connection

Docker creates its isolated docker network in which container can communicate ,when I declare two container is same network they can communicate. With just the container name.



If application is running outside the container it will connect via host name and port no



Docker network ls

A black screen with white text

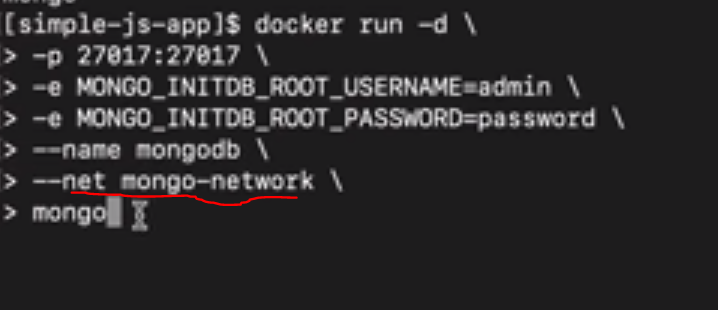
Description automatically generated

If we want to create our own

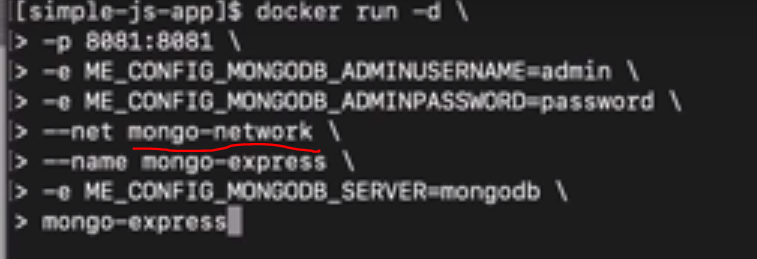
A computer screen with white text

Description automatically generated

Now run docker image with network



Now we want to start express and it should connect to mongo db



# Push Images on ECR

Docker images (To list)

You have to login first

Docker login (incase of aws check login command in ECR)