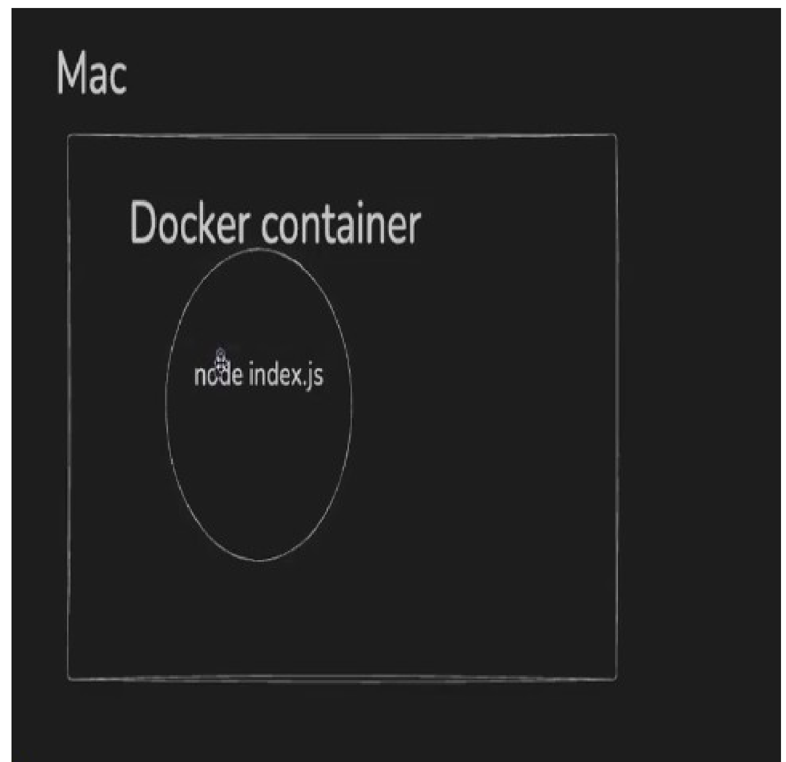
DOCKER CONTAINER === ITS LIKE A SMALL MACHINE
LIKE A VM WE CAN SAY
ONE USECASE IS THIS LIKE WHEN YOUR FRIEND SOMETHING SEND YOU SUS AND YOU WANT TO TRY WHAT IS THAT RUN IT INSIDE THE DOCKER CONTAINER
THEY ALSO HAVE THERE OWN FILESYSTEM



Docker/containers are important for a few reasons -

1. Kubernetes/Container orchestration
2. Running processes in isolated environments
3. Starting projects/auxilary services locally

EG: postgress, kafka, redis ets and projects like some github projects also has manual installation and docker installation both like the p5 js web editor and just a single command can run a full project dammmmmn

Step 2 - Containerization

What are containers

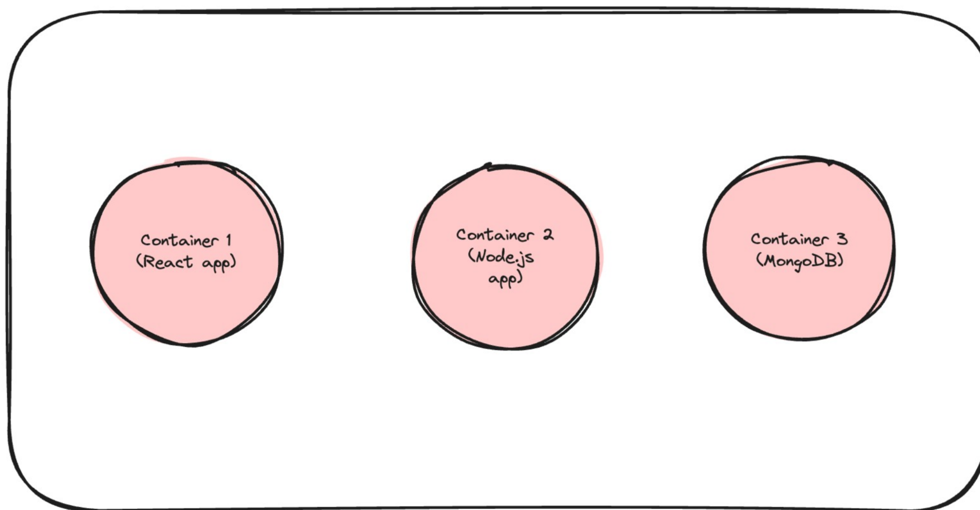
Containers are a way to package and distribute software applications in a way that makes them easy to deploy and run consistently across different environments. They allow you to package an application, along with all its dependencies and

libraries, into a single unit that can be run on any machine with a container runtime, such as Docker.

Why containers

1. Everyone has different Operating systems
2. Steps to run a project can vary based on OS
3. Extremely harder to keep track of dependencies as project grow

Benefits of using containers



1. Let you describe your configuration in a single file
2. Can run in isolated environments
3. Makes Local setup of OS projects a breeze
4. Makes installing auxiliary services/DBs easy

References

- For Reference, the following command starts mongo in all operating systems -

```
docker run -d -p 27017:27017 mongo
```

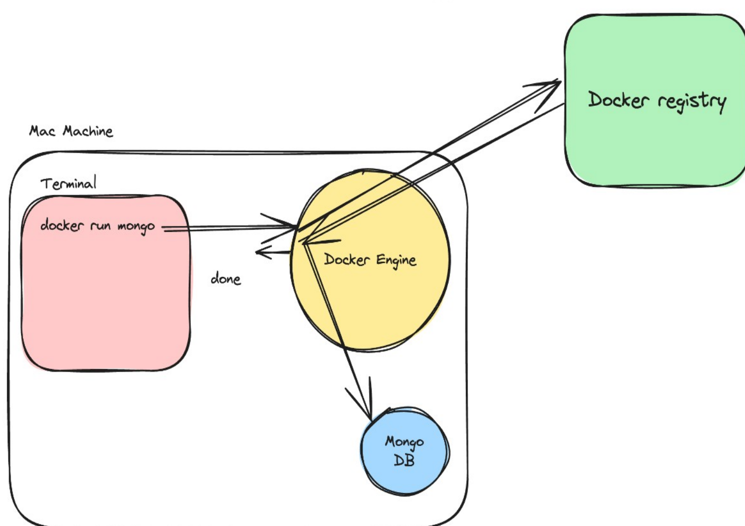
- Docker isn’t the only way to create containers

Installing docker

Download the docker cli and install it

then try to run the docker

Inside docker



As an application/full stack developer, you need to be comfortable with the following terminologies -

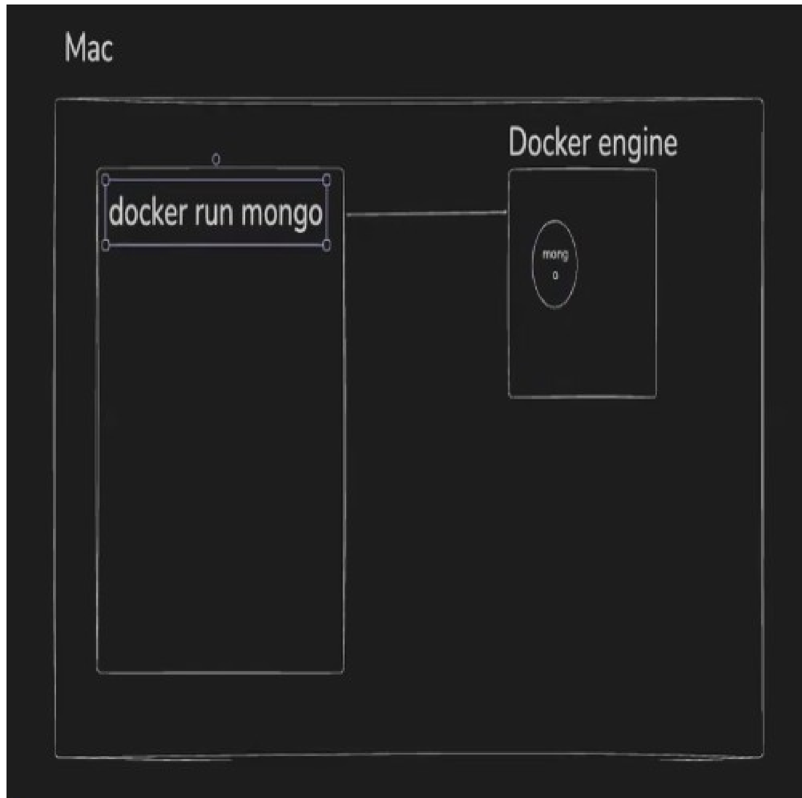
1. Docker Engine
2. Docker CLI - Command line interface
3. Docker registry

1. Docker Engine

Docker Engine is an open-source containerization technology that allows developers to package applications into container

Containers are standardized executable components combining application source code with the operating

system (OS) libraries and dependencies required to run that code in any environment.



2. Docker CLI

The command line interface lets you talk to the docker engine and lets you start/stop/list containers

```
docker run -d -p 27017:27017 mongo
```

Docker cli is not the only way to talk to a docker engine. You can hit the docker REST API to do the same things

3. Docker registry

The docker registry is how Docker makes money.
It is similar to github, but it lets you push images rather than sourcecode
Docker's main registry - https://dockerhub.com/
Mongo image on docker registry -
https://hub.docker.com/_/mongo

WHAT IS REGISTRY?

ITS A PLACE WHERE WE PUSH THE CODE LIKE FOR NPM WE STORE THE CODE, GITHUB IS ALSO LIKE A REGISTRY AND PIP IS REGISTRY FOR PYTHON ALSO

DOCKER REGISTRY

This contains all the code for the packages like for express, mongo etc..

https://hub.docker.com/_/python

try

docker run mongo

try using with mongodb compass

docker run -p 27017:27017 mongo (for a specific port)

docker images(to check the installed images)

```
Cohort\Dev\week-26-docker took 1m59s
❯ docker images
                                                  i Info →   U  In Use
IMAGE           ID             DISK USAGE   CONTENT SIZE   EXTRA
mongo:latest    474f5c3bf0e3       1.29GB          338MB     U

Cohort\Dev\week-26-docker
```

```
Cohort\Dev\week-26-docker
❯ docker rmi 474f5c3bf0e3
Error response from daemon: conflict: unable to delete 474f5c3bf0e3 (must be forced) - image is being
used by stopped container 38cbf4d3b7d4

Cohort\Dev\week-26-docker
❯ docker rm 38cbf4d3b7d4
38cbf4d3b7d4

Cohort\Dev\week-26-docker
❯ docker rmi 474f5c3bf0e3
Error response from daemon: conflict: unable to delete 474f5c3bf0e3 (must be forced) - image is being
used by stopped container cc68a1b77425

Cohort\Dev\week-26-docker
❯ docker rmi 474f5c3bf0e3 --force
Deleted: sha256:474f5c3bf0e355bb97dafda730e725169a4d51c5578abf7be9ec7ad3fdee4481
```

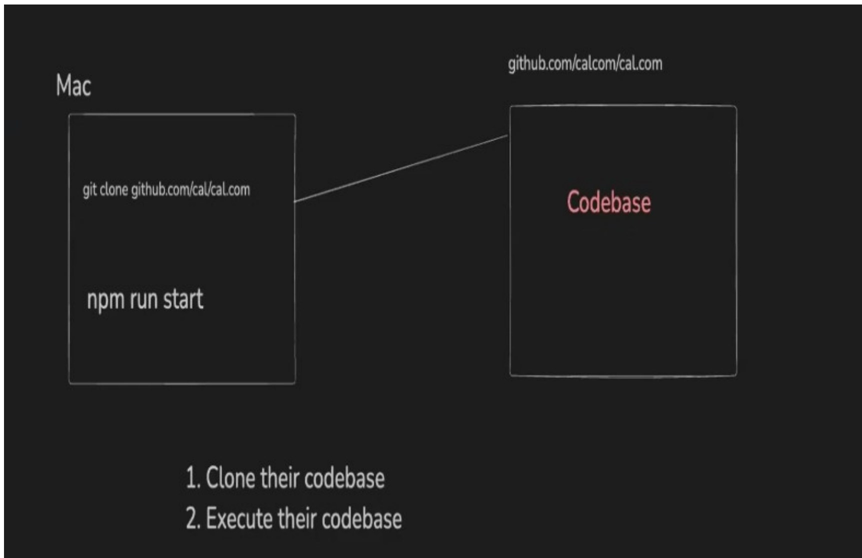
use the id to delete the docker from the list

rm == stops the instance

rmi == delete the instance we need to force also

docker ps == to check the total number of process running

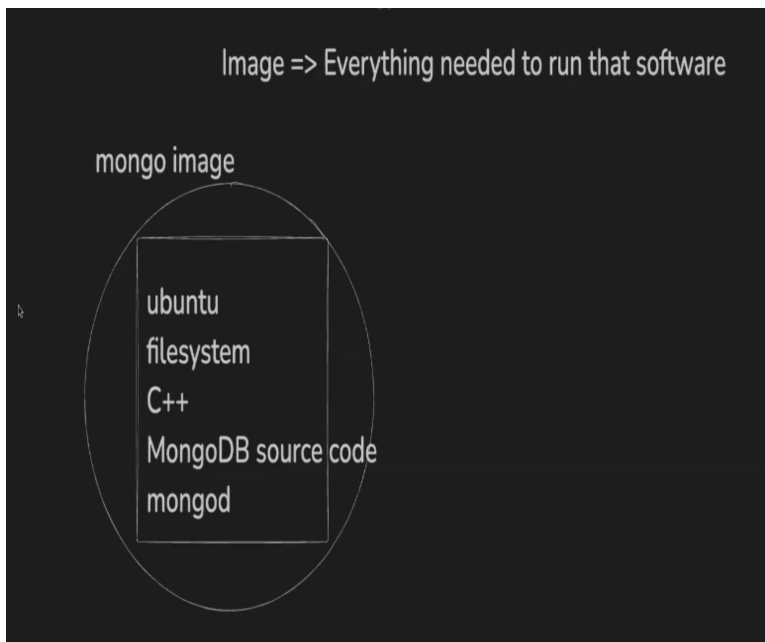
docker kill id__ ==== kills the process



what we used to do earlier to run a project

IMAGE == the thing we are pulling from hub.dockerhub.com is an image eg: mongodb, express

ITS LIKE CLONING A CODEBASE



the image conatain all this things the code, source code etc ...

CONTAINER == Image in execution is called as container

ITS LIKE RUNNING A CODE BASE

when we run this docker run -p 27017:27017 mongo

this starts a container and when we stop it its called as stopping container

Images vs containers

Docker Image

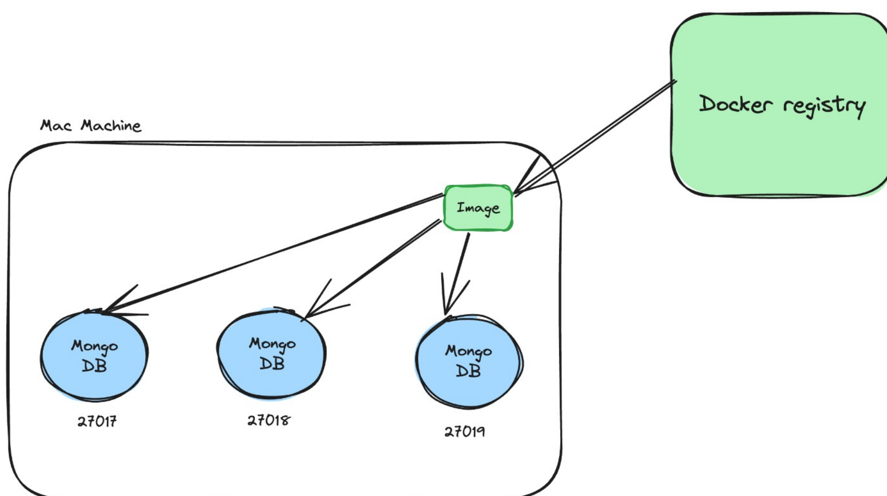
A Docker image is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.

A good mental model for an image is Your codebase on github

Docker Container

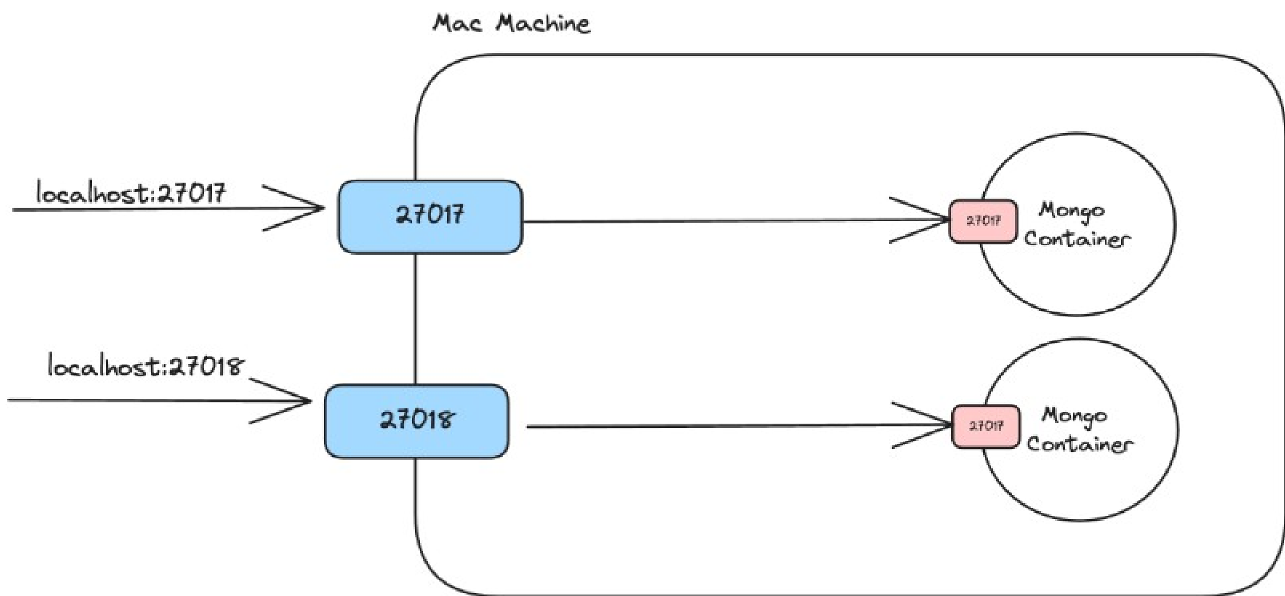
A container is a running instance of an image. It encapsulates the application or service and its dependencies, running in an isolated environment.

A good mental model for a container is when you run node index.js on your machine from some source code you got from github



Step 7 - Port mapping

```
docker run -d -p 27018:27017 mongo
```



in this its like the port for the container is defined bytthe maker and also its like if we have same container port but running on different system port so they are different they are not the same

like

docker run -p 27017:27017 mongo

docker run -p 27018:27017 mongo

these both will have different database and also will run on two different ports

first port == machine port

second port == container port

WHEN WE RUN THE COMMAND WITHOUT THE PORTS THEN THE ISSUE IS THE PORT IS JUST EXPOSED IN THE VIRTUAL MACHINE NOT THE HOST MACHINE

TO EXPOSE THAT TO HOST MACHINE YOU HAVE TO DEFINE THE PORT TO ACCESS

1. docker images

Shows you all the images that you have on your machine

2. docker ps

Shows you all the containers you are running on your machine

3. docker run

Lets you start a container

-p ⇒ let's you create a port mapping

-d. ⇒ Let's you run it in detatched mode

4. docker build

Lets you build an image. We will see this after we understand how to create your own Dockerfile

docker build is used for creating your own specific docker files

5. docker push (for the makers of the dockerfile to push the code to repo)

Lets you push your image to a registry

6. Extra commands

docker kill

docker exec

most used command

docker exec -it container-id sh

it == interactive mode (its like the terminal opens of that container)

to run the container

```
docker run -d -p 27017:27017 mongodb:4.7-replset
```

to get the container id

```
docker ps
```

to get into the terminal

```
docker exec -it 16236517659e sh
```

sh == shell script(used for debugging the dockerfile)

while making a project its good to make a container as its good for deployment and also like if you want to use container orchestration you need to containrize first to do that

HOW TO CONTAINERIZE A NODE JS APPLICATION??

this same task was given to a $20k person with like a vm with 6 folders with all of them simple code so what you have to do like create dockerfile for each of them

express process code also

Creating a docker file

its like a config file where we define everything require to run that project

will contain all the commands which is required to run the project

node-alpine lighter version of nodejs

What is a Dockerfile

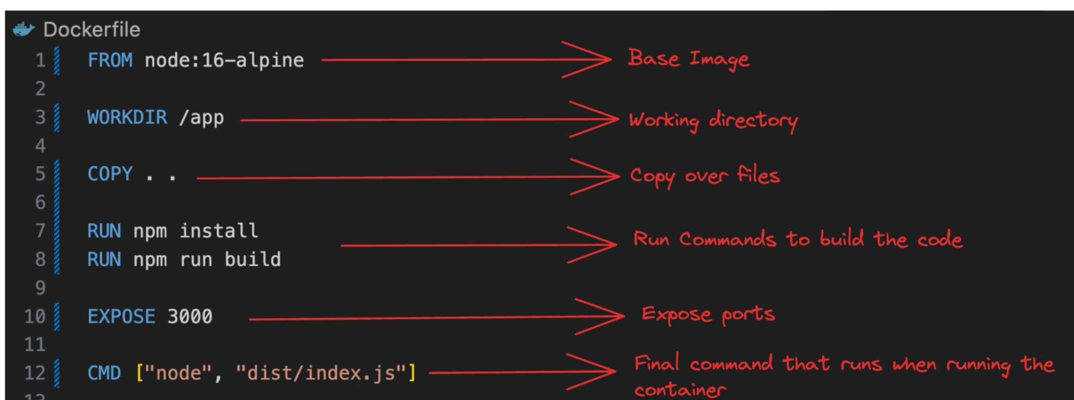
If you want to create an image from your own code, that you can push to dockerhub, you need to create a Dockerfile for your application.

A Dockerfile is a text document that contains all the commands a user could call on the command line to create an image.

How to write a dockerfile

A dockerfile has 2 parts

1. Base image
2. Bunch of commands that you run on the base image (to install dependencies like Node.js)



in this the first line is the base image like can be node, mongodb

COPY .. to copy all the folder to the folder ./app

THIS HAS A FLAW LIKE IT WILL ALSO COPY THE NODE_MODULES FOLDER

to fix this we create .dockerignore in which add node_modules to not copy this folder

CMD = jabh container shtart hua toh kya start kare and also it just can be one file only

so when we run the docker run -p 3000:3000 pdproject

then just the cmd command is run

Common commands

WORKDIR: Sets the working directory for any RUN, CMD, ENTRYPOINT, COPYinstructions that follow it.

RUN: Executes any commands in a new layer on top of the current image and commits the results.

CMD: Provides defaults for executing a container. There can only be one CMD instruction in a Dockerfile.

EXPOSE: Informs Docker that the container listens on the specified network ports at runtime.

ENV: Sets the environment variable.

COPY: Allow files from the Docker host to be added to the Docker image

when we build the image other than cmd all the commands works

and when we are starting container then only cmd

docker build -t hello-world-app . ⟹ to run build the docker file you made

-t ⟹ tag of the app

now see in the docker images it will show and now rather than doing npm run dev you can just run docker command

docker run -p 3000:3000 hello-world-app

docker kill a067fc5573e4 to fully stop it

project can be used without installing node.js

Orchestration is the automated coordination, management, and sequencing of multiple tasks, systems, or services to achieve a complex, end-to-end outcome (Kubernetes)

Passing a env variable with the docker command

addded this to code

app.listen(3000);

console.log("ENV VARIABLE")

console.log(process.env.DATABASE_URL)

docker run -p 3000:3000 -e DATABASE_URL=pd@123 hello-world-app

OUTPUT:

ENV VARIABLE

pd@123

you can pass the env data with the docker file but not with the code as that will be pushed to github

More commands

docker kill = to kill a container

docker exec = to exectue a command inside a container

Examples

1. List all contents of a container folder

docker exec <container_name_or_id> ls /path/to/directory

2. Running an Interactive Shell

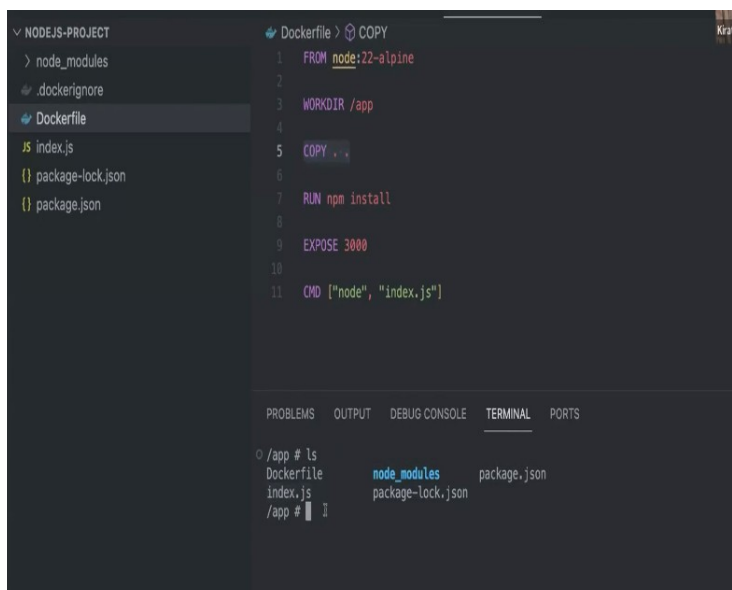
docker exec -it <container_name_or_id> /bin/bash

docker ps (check if the dockerfile is running or not)

if not then try to run the file using the port command

docker exec -it bf94cfff0888 sh

you cannot run the/bin/bash command on the alpine image



but when you run the sh command it contain all the project files

copy command worked as expected

Pushing to dockerhub

Create repo from the website or from the cli like this

first do docker login

docker build -t pdgamersg/basic-devops .

docker push pdgamersg/basic-devops

and by default its always public

docker run -p 3000:3000 pdgamersg/basic-devops

now anyone can run this

as this is a public repo thats why

DOPPLER used to store the env variables

argon alternative to bcrypt

leetcode clone using dockerhub

DOCKER IS LIKE EASY WAY OF DEPLOYING PROJECTS AND ALSO LIKE RUNNING THE PROJECT VERY FAST, HELPS IN MOVES ACROSS MACHINE SUCH AS EC2 AND ALL

YOU CAN RUN ON ANY OS

ALSO PROVIDES ISOLATION FROM THE HOST MACHINE

CONTAINERS ARE VERY LIGHT WEIGHT AS COMPARE TO VIRTUAL MACHINES AND ALL

YOU CAN ALSO SEND HTTP REQUEST WITH THE DOCKER

LAYERS IN DOCKER

```dockerfile
FROM node:25.8.2-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

CMD ["node","index.js"]
```

Best use of this is for **Caching** INTERVIEW QUESTION WHY ITS USED?

THESE FIRST FOUR LINES ARE THE LAYERS OF THE DOCKERFILE

In four commands if till third command is cached then the above those command will be cached

```
fb6cb4bf0fdfb9aa8ce  0.0s
 => [internal] load build context
                      0.0s
 => => transferring context: 315B
                      0.0s
 => CACHED [2/4] WORKDIR /app
                      0.0s
 => [3/4] COPY . .
                      0.0s
 => [4/4] RUN npm install
                      1.1s
 => exporting to image
```

docker build | docker build
wodkdir /app | wodkdir /app
node | node
index.js | index.js
res.send("hello world"); | res.send("hello world 2")

why the third layer was uncached in this ?

Because the code changed in this and to fix that the code has to be cached again

in this first many layers are created then by merging that iso is created

OPTIMIZING THE DOCKER FILE (npm install)

now the issue is run npm install is a command inside the file which is a heavy command to run

as the pacakge.json does not change so often so this can be fixed how?

By creating a different layer for cashing the npm install and will be used only when the package.json is changed

as we are just changing the source code only then why are we running this again and again

```
1  FROM node:25.8.2-alpine
2
3  WORKDIR /app
4
5  COPY ./package.json ./package.json
6  COPY ./package-lock.json ./package-lock.json
7  RUN npm install
8
9  COPY . .
10
11 EXPOSE 3000
12
13 CMD ["node","index.js"]
```

see in this the to cache the npm install we moved it up and also the npm install cannot be used without package.json so to fix that to we use this approach

and now if we make changes to the index.js file the npm install will be cached

Case 1 — You change your source code (but nothing in package.json/prisma)

```
Dockerfile
1  FROM node:20                      Layer 1 - Cached
2
3  WORKDIR /usr/src/app              Layer 2 - Cached
4
5  COPY package* .                   Layer 3 - Cached
6  COPY ./prisma .                   Layer 4 - Cached
7
8  RUN npm install                   Layer 5 - Cached
9  RUN npx prisma generate           Layer 6 - Cached
10
11 COPY . .                          Layer 7 - UnCached
12
13 EXPOSE 3000                       Layer 8 - UnCached
14
15 CMD ["node", "dist/index.js", ]
```

Case 2 - You change the package.json file (added a dependency)

```dockerfile
FROM node:20                                   # Layer 1 - Cached

WORKDIR /usr/src/app                           # Layer 2 - Cached

COPY package* .                                # Layer 3 - UnCached
COPY ./prisma .                                # Layer 4 - UnCached

RUN npm install                                # Layer 5 - UnCached
RUN npx prisma generate                        # Layer 6 - UnCached

COPY . .                                       # Layer 7 - UnCached

EXPOSE 3000                                    # Layer 8 - UnCached

CMD ["node", "dist/index.js", ]
```

VOLUMES

Currently its like whenever you run a docker file and then add some data and stop it the data is lost

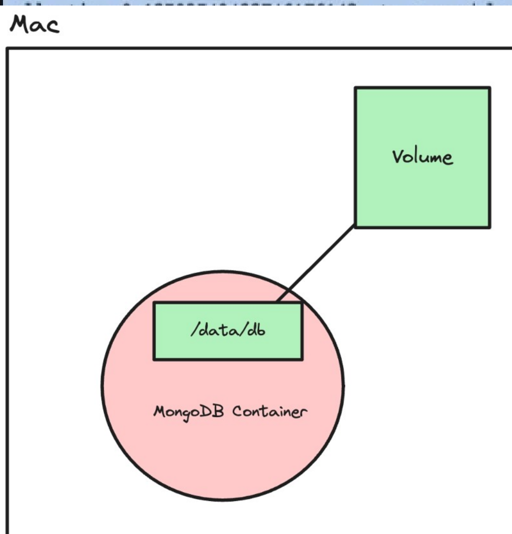
DOCKER ARE STATELESS They should not persist data
DOCKER ARE TRANSITIVE

now if we want like start the database and we want to persist the data what we should do?

```
→ ~ docker exec -it 42122a5d6beb sh
# cd /data/db
# ls
WiredTiger            diagnostic.data
WiredTiger.lock       index-1-1250354942274617б142.wt
WiredTiger.turtle     index-3-1250354942274617б142.wt
WiredTiger.wt         index-5-1250354942274617б142.wt
WiredTigerHS.wt       index-6-1250354942274617б142.wt
_mdb_catalog.wt       journal
```

For mongodb the data is stored here we have to somehow copy this to persist data and this type of location for saving data is not same for each db

so to like save this data we need to mount this data somewhat outside this is for volume is used for



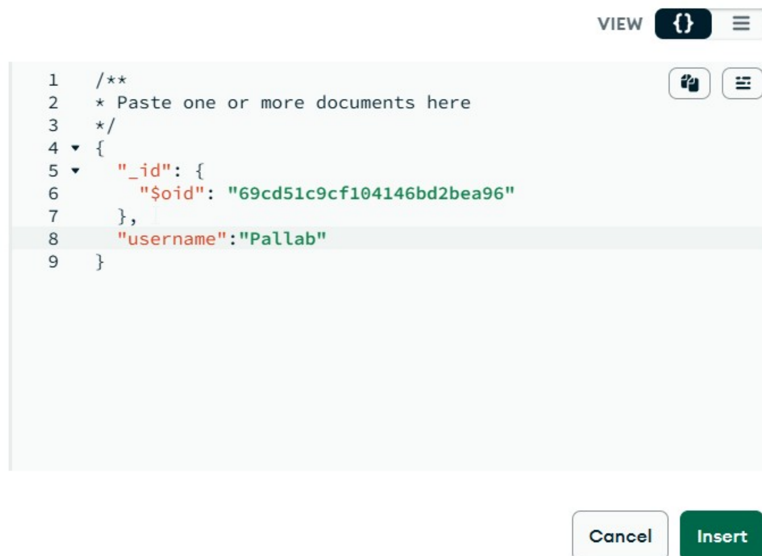
first to use this create a volume

docker volume create mongo_db_data

now I want the changes of /data/db should be sent to volume folder the state get persist outside the containter

docker run -d -p 27017:27017 -v mongo_db_data:/data/db mongo

adding data in mongo connecting to connection then creating new todo_app with users as folder



adding entry

now after adding this if you kill the db using docker kill db_id

then check the docker volume ls

there the data still exist

and if we run without the volume location then nothing will be seen

https://github.com/100xdevs-cohort-2/week-15-live-2.2

using this repo and starting mongo server

changed url to localhost of the db

npm run build

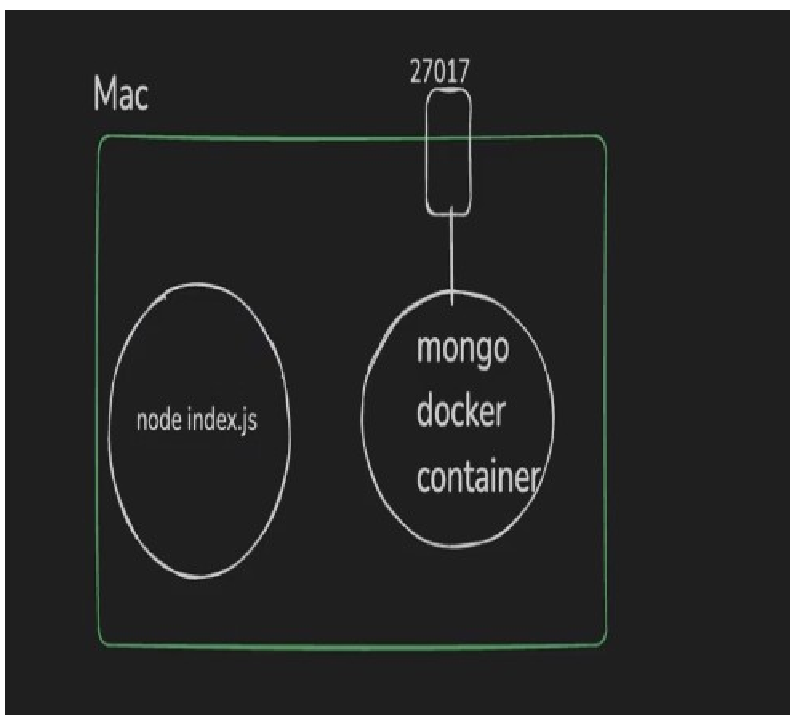
node dist/index.js

http://localhost:3000/user

sending data using the body using postman

```
{
  "name":"pd",
  "age":21,
  "email":"hello@gmail.com"
}
```

and is shown in the db



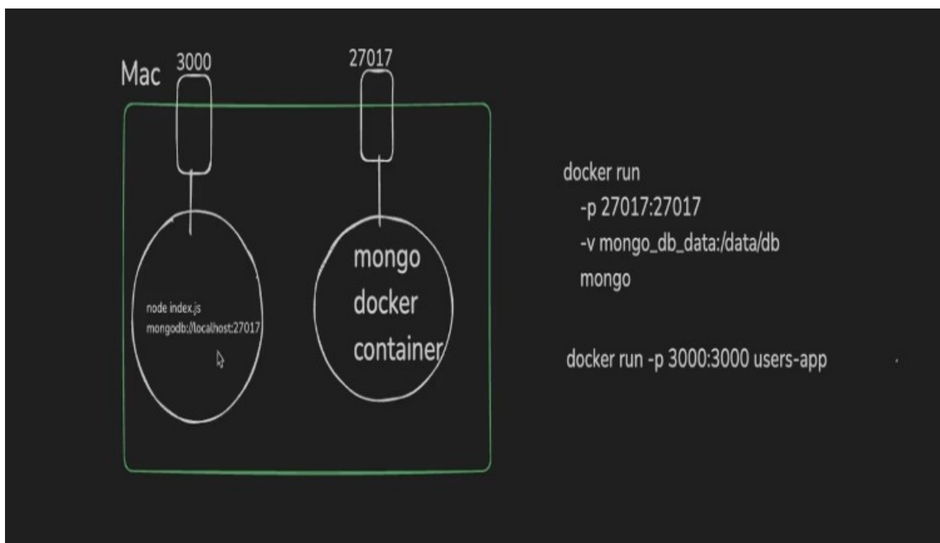
for now we are runnign the node index.js outside not inside a docker so trying to do that now

NOW BUILDING THE TS PROJECT

docker build -t users-app .

docker run -p 3000:3000 users-app

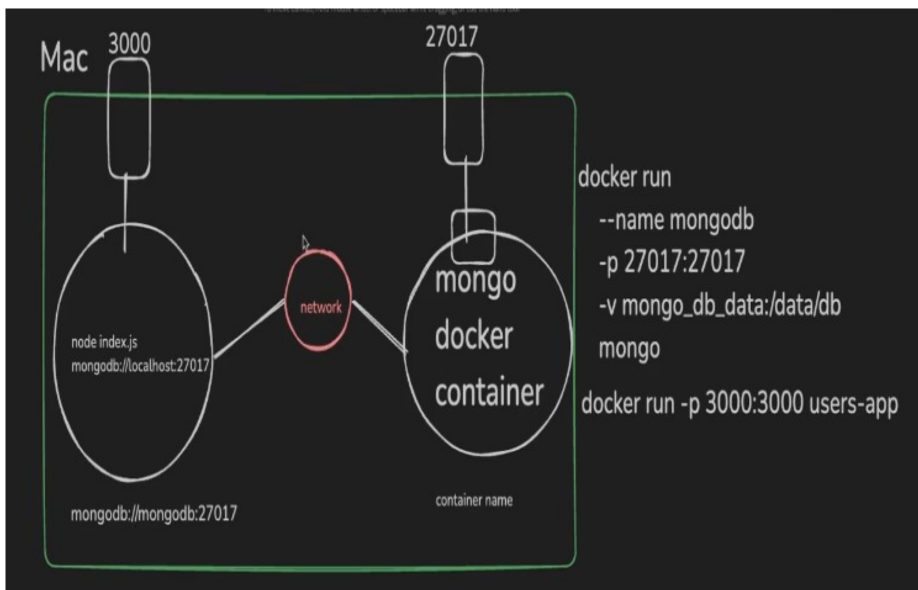
now the project is running but not showing the db connection ??



as in this the node module has its own ports and the db is working on the exposed on the mac machine ip not the container thats why the node process is not able to access the port , the machine in which the node process is working in that the mongodb is not working

SO HOW CAN ONE CONTAINER TALK TO THE OTHER CONTAINER

THIS IS CALLED AS NETWORK



first we have to create the network

so the network is like which can connect the two containers how? First give name to the containers and then that name can be used in the url instead of the localhost we can use the container name

HOW TO CREATE A NETWORK?

docker network create node_app_network

docker network ls

docker run --name mongodbcontainer --network node_app_network -p 27017:27017 -v mongo_db_data:/data/db mongo

docker build -t users-app .(to get the updated data)

added the network in this now while running and the name will be used in the place of localhost

docker run --network node_app_network -p 3000:3000 users-app

now its mongodb connected

INSTEAD OF CREATING A DOCKER FILE WE CAN ALSO CREATE A DOCKER COMPOSE

docker rmi file_name to remove the docker image

DOCKER COMPOSE

learning docker-compose

monorepo deployment to a VM via CI/CD using docker

HELPS TO RUN MULTIPLE DOCKER FILES TOGETHER

```
~ docker run 100xdevs/dailycode-staging
~ docker build -t frontend
~ docker build -t backend
~ docker run mongo
```

the docker compose is used for basically saving time in running multiple things like mongo, postgress, building the projects fronted backend etc

so to fix that we like create this but its used mainly for the local development not in the deployment because where we use actual db not the docker db

like we create volume to persist the data because as we kill the db the data is lost so the volume keeps the data of the db and for mongo its stored in the /data/db so in this compose file also we create the volume for the image we are running

```yaml
# docker-compose.yaml
version: '3.8'
services:
  mongodb:
    image: mongo
    container_name: mongodb
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db

  backend22:
    image: backend22
    container_name: backend_app
    depends_on:
      - mongodb
    ports:
      - "3000:3000"
    environment:
      MONGO_URL: "mongodb://mongodb:27017"

volumes:
  mongodb_data:
```

Version of docker-compose
Mongo service
Backend service
Volumes to create

this is the explanation

two services are running just way of writing is different

docker run mongo -name mongodb -p 27017:27017-v mongodb_data:/data/db

these both services just run when I type

docker-compose up

network is also used in this

Initializing the project

npm init -y

npx tsc --init

change the dir also

create the src folder with index.ts

npm install prisma express @types/express

npx prisma init

"type":"module"

docker run -e POSTGRES_PASSWORD=mysecretpassword -d -p 5432:5432 postgres

in this the

-e == env file

-d == detach mode

-p == port

DATABASE_URL="postgresql://postgres:mysecretpassword@localhost:5432/postgress"

change this url in the .env file also

what you can also do get the url from the neondb also and paste it here

npx prisma migrate dev

npx prisma generate(to generate the client)

docker rm project if issue

```
15  ##Docker installation
16  - Install docker
17  - Creat a network - docker network create user_project
18  - Start postgres
19      - docker run --network user_project --name postgres -e
        POSTGRES_PASSWORD=mysecretpassword -d -p 5432:5432 postgres
20  - Build the image - `docker build --network=host -t user-project .`
21  - Start the image - `docker run -e DATABASE_URL=postgresql://
    postgres:mysecretpassword@postgres:5432/postgres --network user_project -p
    3000:3000 user-project`
22
23  ##Docker Compose installation steps
24  - Install docker, docker-compose
25  - Run `docker-compose up`
```

Earlier the docker file was having some issues like we have to create the network by ourself but not now and also there were so many steps

DOCKER COMPOSE YML FILE

```
TS index.ts | schema.prisma | Contribute.md | Dockerfile | docker-con

1   version: '3.8'
    ▷Run All Services
2   services:
      ▷Run Service
3     postgres:
4       image: postgres
5       ports:
6         5432:5432
      ▷Run Service
7     user_app:
8       build:
9         context: ./
10        dockerfile: Dockerfile
11      environment:
12        DATABASE_URL="postgresql://postgres:mysecretpassword@localhost:5432/
          postgress"
13      ports:
14        3000:3000
15      depends_on:
16        -postgres
```

in this the network is automatically created and also looks clean

make sure while creating look at the indentation as that matters

in this the run service comes on the top so look at that will starting the service

when we add - this dash it becomes a list

to remove the docker-compose up cache

docker images (get the top name it will be folder name)

docker rmi image_name --force

docker compose down && docker compose up -build(emergency command)

UNDERSTANDING NOW

```
##Manual installation
 - Install nodejs locally ()
 - Clone the repo
 - Install dependencies (npm install)
 - Start the DB locally
    - docker run -e POSTGRES_PASSWORD=mysecretpassword -d -p 5432:5432 postgres
    - Go to neon.tech and get yourself a new DB
- Change the .env file and update your DB credentials
- npx prisma migrate
- npx prisma generate
- npm run build
- npm run start


##Docker installation
- Install docker
- Creat a network - docker network create user_project
- Start postgres
    - docker run --network user_project --name postgres -e POSTGRES_PASSWORD=mysecretpassword -d -p 5432:5432 postgres
- Build the image - `docker build --network=host -t user-project .`
- Start the image - `docker run -e DATABASE_URL=postgresql://postgres:mysecretpassword@postgres:5432/postgres --network user_project -p 3000:3000 user-project`

##Docker Compose installation steps
- Install docker, docker-compose
- Run `docker-compose up`
```

the above things are the instructions to be given and it reduces as the method changes

```dockerfile
FROM node:20-alpine

WORKDIR /app

COPY ./package.json ./package.json
COPY ./package-lock.json ./package-lock.json

RUN npm install

COPY . .

RUN npx prisma generate
RUN npm run build

EXPOSE 3000

CMD ["npm","run","dev:docker"]
```

this I fairly understand but

what is this dev:docker

this is defined in the package.json

```
"scripts": {
    "build": "tsc -b",
    "dev:docker": "npx prisma migrate deploy && node dist/src/index.js"
  },
```

this was defined like this because the issue was the postgress was fucking irritating and was unable to run during the docker file build time so we moved to the run time only

index.ts | schema.prisma | docker-compose.yml | tsconfig.json

```yaml
version: '3.8'
services:
  postgres:
    image: postgres
    ports:
      - 5432:5432
    environment:
      - POSTGRES_PASSWORD=mysecretpassword
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 10

  user_app:
    build:
      network: host
      context: ./
      dockerfile: Dockerfile

    environment:
      - DATABASE_URL=postgresql://postgres:mysecretpassword@postgres:5432/postgres
    ports:
      - 3000:3000
    depends_on:
      postgres:
        condition: service_healthy
```

see this is a docker compose file which is like docker file but we can add anything to make it run like db and the commands all of them

its easy to understand nothing to completed just learn the format

the health checks and all were added by claude