19/9/24

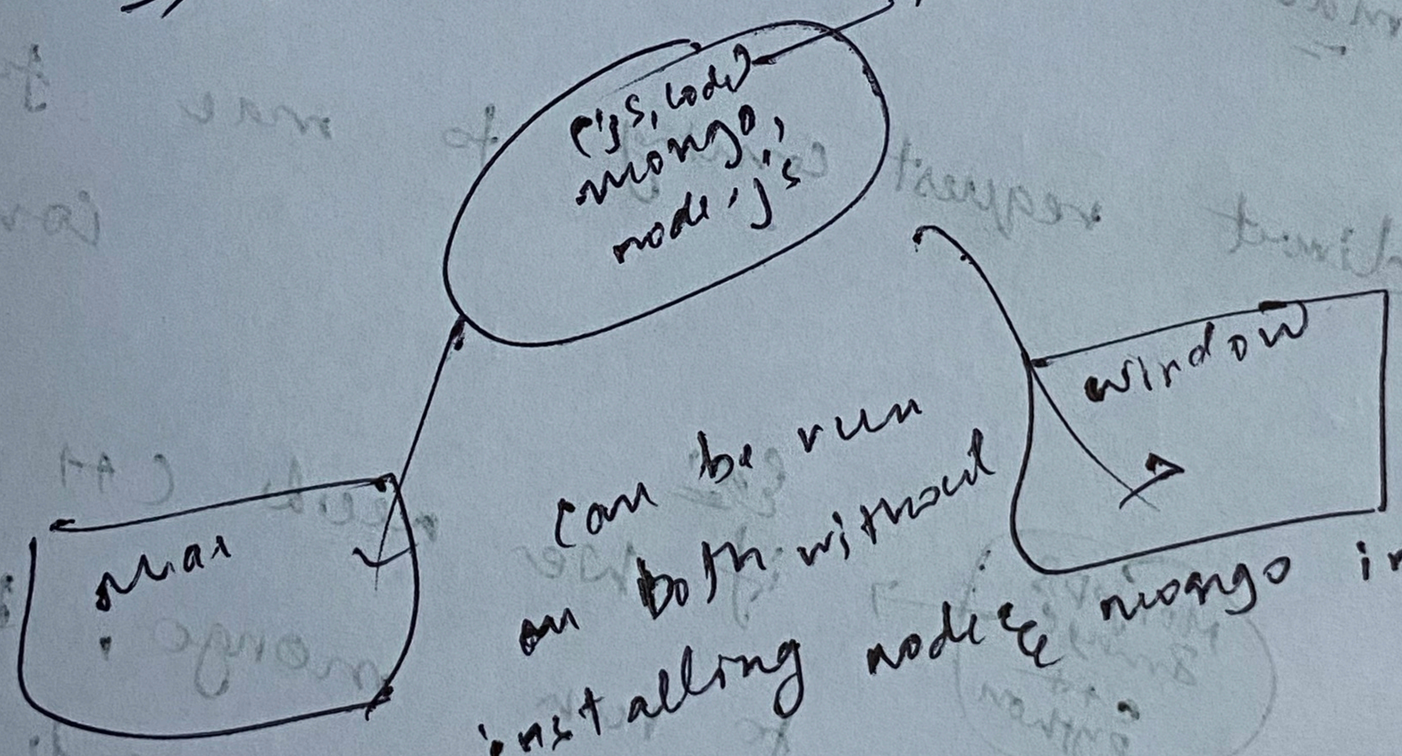
Docker

→ Containerization:

Mini machine inside a machine (main).

→ 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.

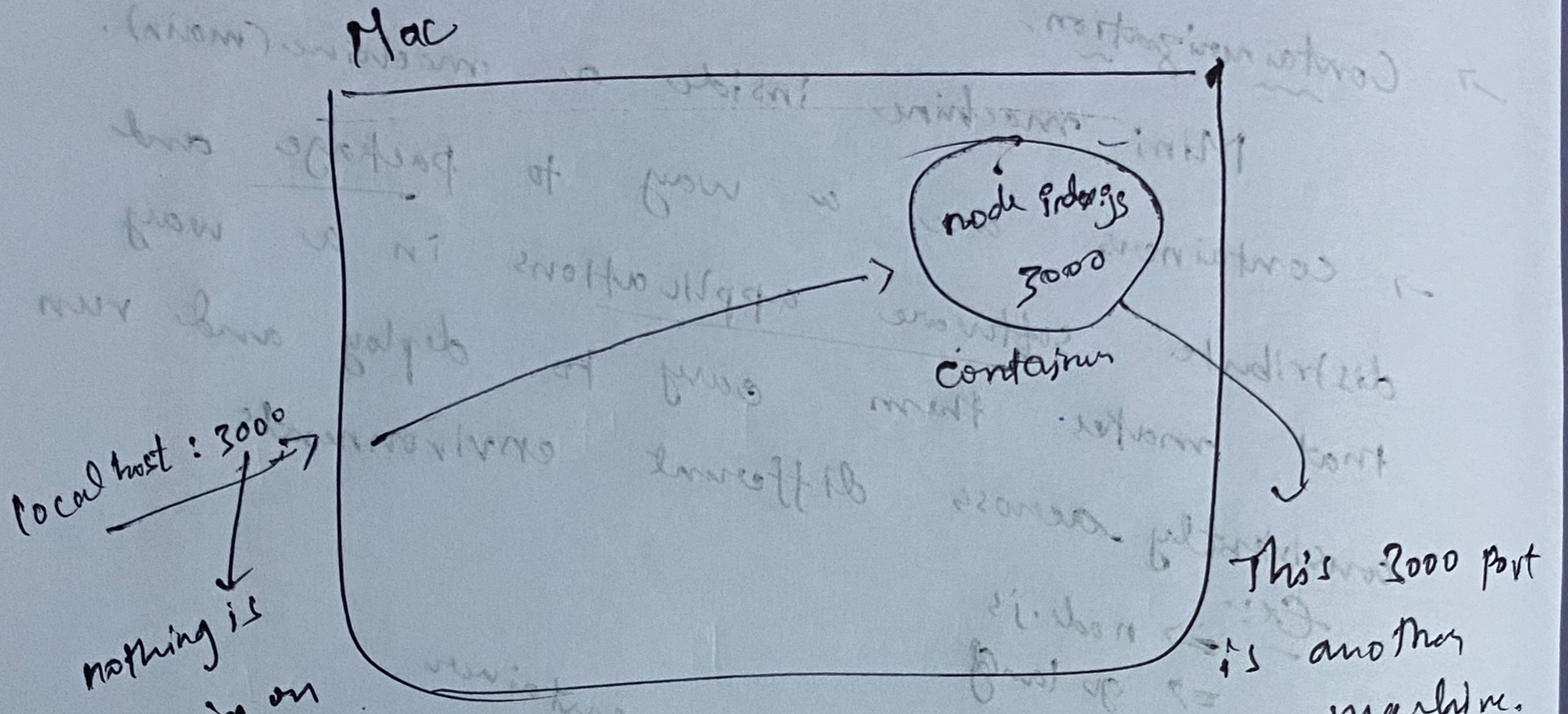
Ex:-
- => node.js
- => go lang
- => rust.



* Docker isn't the only way to create containers.

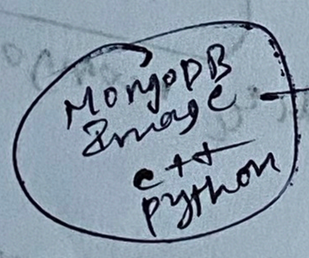
Actionable Docker:-

→ Let's see how to run packages locally.



→ Can direct request coming to mac to the containers.

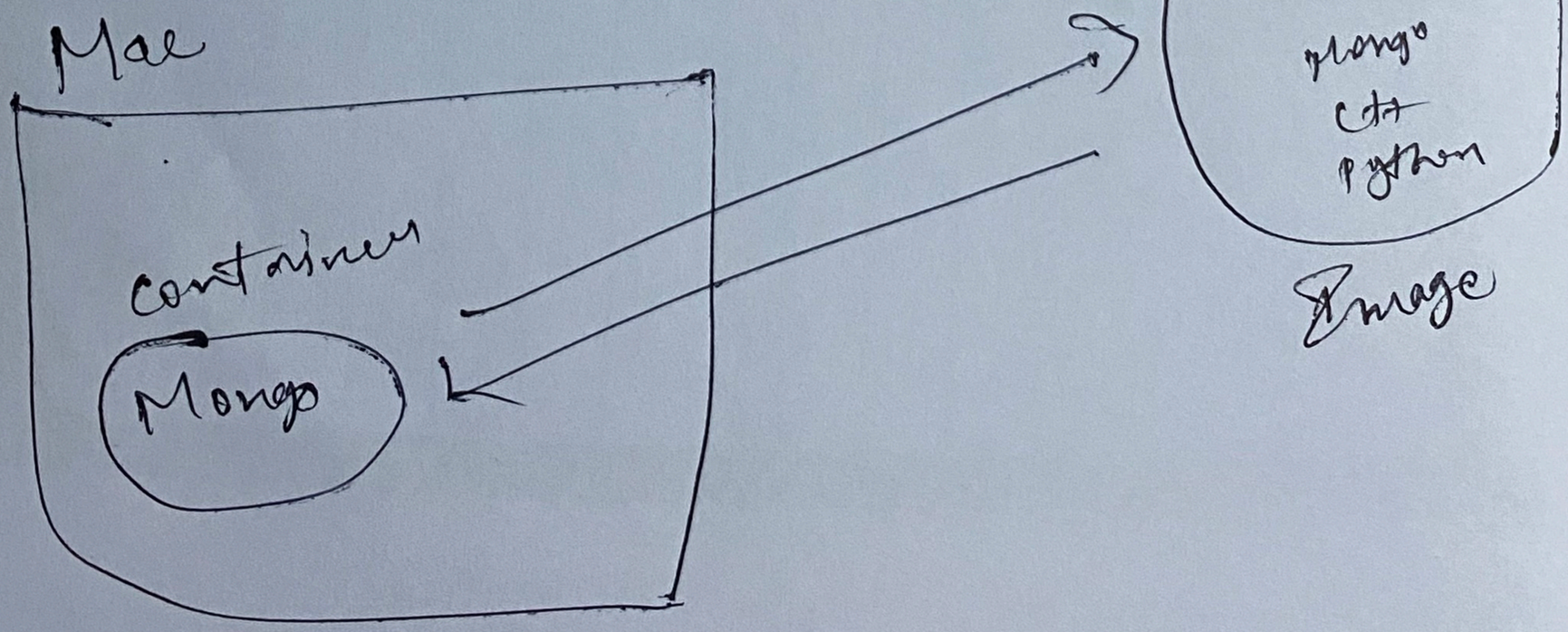
* Image :-



Image

Ex: if we need C++, Python to run, mongo: bundle everything it needs to run.

Container → Image in Execution



→ Commands:.

1. docker pull mongo. → pull image doesn't start
2. docker run mongo → to start container (if Image doesn't exist it will pull & run).

Port mapping:.

docker run -p 27017:27017 mongo

27017
xxxxx → can be any port of mac

run in detached mode | Background:-

docker run -d -p 27017:27017 mongo.

Stopping Containers:

docker kill <container-id>.

List running containers:.

docker ps

Postgres:.

docker run -e → environment variable. POSTGRES_PASSWORD=mysecretpassword -d -p 5432:5432 postgres

Then Connection string of postgres would be

postgresql://postgres:mysecretpassword@localhost:5432/postgres.

→ docker images (List all images)

→ docker rmi <image_id> (delete image)

→ docker rm <container_id> (delete container).

(Build)

→ docker build -t <name> . → path of Dockerfile.

(SSH)

→ docker exec -it <container_id> /bin/bash

passing env variables:-

→ docker run -p 3000:3000 -e DATABASE_URL="..." <image>;

22/9/24

* Layers in Docker:-

→ Docker shares layers across images.

(cached)

⇒ 1. Base image creates first layer.

2. Each RUN, COPY, WORKDIR command creates new layer.

→ Layers can get re-used across docker builds.

(We can reduce build time by caching)

Ex:-

```
FROM node:20

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

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

Here, npm install only depends on package.json.

But COPY . . means whenever there is change in any file it will be uncached.

(We rarely change package.json)

→ optimise

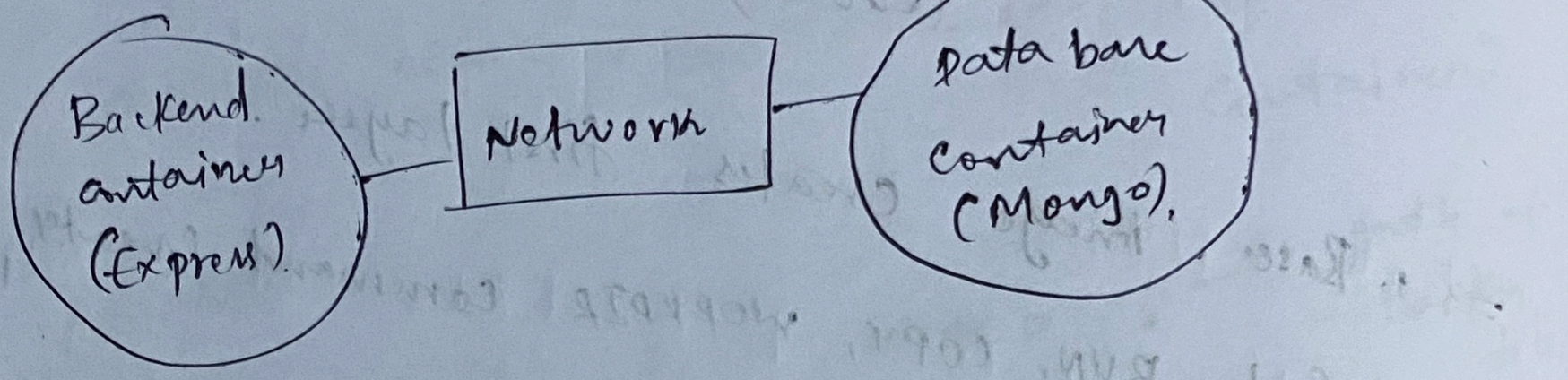
```
FROM node:20

WORKDIR /app

COPY package* .

RUN npm install

COPY . .

EXPOSE 3000

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

By adding these we are caching npm install

* Networks and Volumes

Need to persist data across docker restarts.

Need to allow containers to talk to each other



* 1. Volumes:-

→ docker volume create <volume-name>.

Ex: docker volume create volume-database.

* Now Mount the folder in mongo which actually stores the data to this volume.

> docker run -v volume-database:/data/db -p 27017:27017 mongo

* here /data/db is the mongo image stores data.

It is different for other database

Some may contains more than one volume.

* 2. Networks:-

→ docker network create <network_name>

Ex:- docker network create my_custom_network.

Now,

1. Start the backend process network attached to it (Express).

→ docker run -d -p 3000:3000 --name backend --network my_custom_network image_tag

2. Start mongo on same network.

→ docker run -d [-v volume-database:/data/db] → optional --name mongo --network my_custom_network mongo

In Express:-

URL = "mongodb://localhost:27017/myDatabase";

replace local host with container name

URL = "mongodb://my_custom_network:27017/myDatabase";

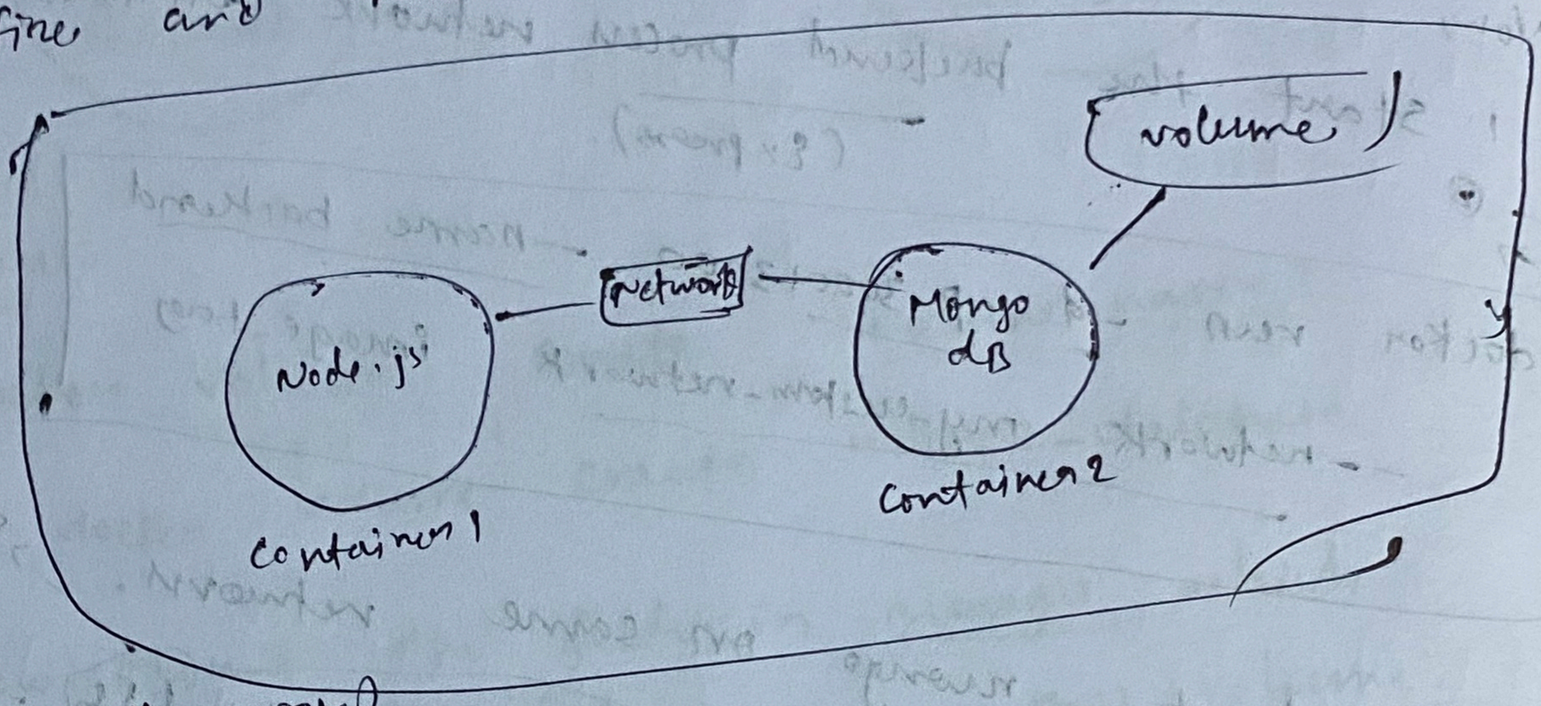
URL = "mongodb://mongo:27017/myDatabase";

23/9/24

* Docker-compose :-

docker-compose.yml

⇒ Docker compose is a tool designed to help you define and run multi-container Docker applications.



(start docker-compose)

→ docker-compose up

---

docker-compose.yml

```
version: '3.8'
services:
  mongodb:
    image: mongo
    container_name: mongodb
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db
  backend2:
    image: backend
    container_name: backend-app
    depends_on:
      - mongodb
    ports:
      - "3000:3000"
    environment:
      MONGO_URL: "mongodb://mongodb:27017"
volumes:
  mongodb_data:
```

* network is created by default

build: .

↓

code don't have image of backend so, we can build it

* Bind Mounts:-

Hot Reloading.

→ docker run -v .:/app -p 3000:3000 react

Current local path

Work dir of container.

⇒ whenever there is change in current local path of src code the /app dir updates in container, so. changes reflects immediately.