Open Source Bootcamp
Docker & Devops

Docker - 1
Hkirat

Simple Node.js app → index.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html')
})

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Containerise this app → (Docker file) Dependencies libraries etc defined

```
FROM node:20
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 3000
CMD ["node", "index.js"]
```

→ img created

besides along with code

① FROM Base-image node:20 → is like base image to run it
20 → specific version

② WORKDIR /usr/src/app
↳ base directory inside image where you want to pull your code; run command

③ COPY . . → copy every thing inside the image from that folder

④ RUN npm install → To create an img; copy source code only → can work

Interview → Should you copy node modules from your file system over to docker image?

Ans) → NO. Copy source code, package.json run npm inside image

⑤ COPY ./index.js . → . . → copy everything
file to be copied

⑤ EXPOSE 3000 → port to be exposed; forwards request

⑥ CMD ["node", "index.js"]
↳ final command to run when container is running

COMMAND to RUN when

CMD & RUN
↓ ↳ doing a bunch of things
RUN without converting to container

Create an img, run it →

docker build . -t. test-app → tag name

↳ Creates image for me

docker images

To run images → CMD →

docker ps

docker images

↳ check for tag

docker run test-app

↳ start the image or container

To stop image

docker run -p 3000:3000 test-app

reques

3000 port of machine forward to 3000 port of container

3000 on machine point to 3000 on container

~~300~~ say 3003:3000 → ~~3000~~ 3003 की requ को 3000 पर

Push code on docker hub

docker push <image-tag>

Create a repo on docker

docker pull <image-tag>

docker run <image-tag>

docker build -t [repo-name]

like   user_name/name-given

docker push [repo-name] → push image from ~~the~~ local machine to dockerhub

copy pull command from website

sudo pull command

sudo docker pull [link]

└── name of repo by default from docker.hub

sudo docker run [link] ← deployed on internet

first stop the container. then re run it with mapping

sudo docker ~~run~~ -p 3000:3000 ~~run~~ [link]

↳ not secure

<u>To login</u>

docker login

Every step is a layer

कुछ भी change होगा ~~then~~ like node 20 को 19 ~~करेंगे~~ करेंगे,
तो sub कुछ according to change होगा

Assignment
1. figure out way to make docker build faster
2. figure out way to light weight image and run cause like the one worked on was like 1 GB something

Assess → Assignment 4 only

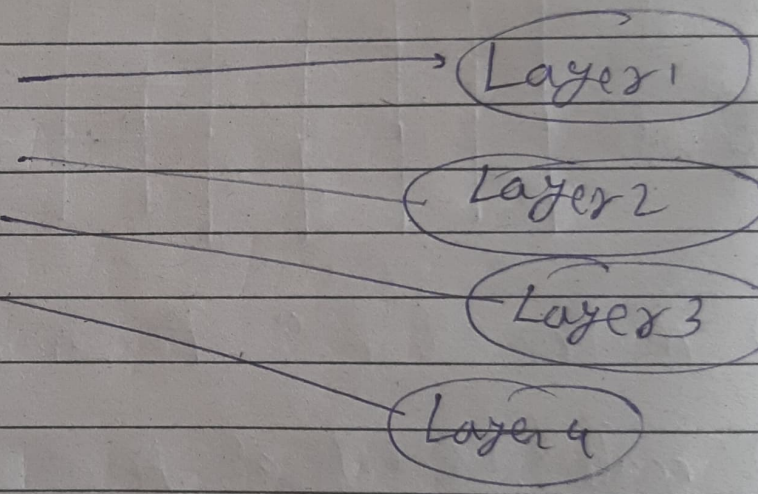
(@3) → Docker - 2

Docker - 2 — HKirat

- Caching images, layers in docker
- Volumes and networks, adding DB
- Multi stage builds
- Docker compose
- open source project

Assignment

```
FROM node:20
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 3000
CMD ["node", "index.js"]
```



Say 3rd layer changed 4, 5 will be changed
1, 2 will be used as it
↳ Caches the layer
for our future use

Layers → Caching
→ Re-using layers
→ faster build times

Say Layer changed

COPY package*.json ⟶ Layer 3

RUN npm install ⟶ Layer 4

COPY . . ⟶ Layer 5

Since already cached hence when re run takes no time to run

Say file changed but the package didn't then what the file with not go again → expensive step but package.json didn't change

script didn't change but the dependencies changed

package.json → doesn't changes very often

index.js is copied into containes at very end

1, 2 ✓

③ → node img; inside node img we have package.json

Layer 4 → node_modules Copy index.js into image

layer 5 rebuild ⟶ layer 4 not rerun

json file 1st copied then to copied over rest

1, 2, 3, 4, → Cached

5 → copied to build

generally js files change hence COPY . . → will run easily

alf tof caching size is like 100 MB docker ruthu thuu 1.1 GB of original
↳ low size; same functionality

Runs on same port 3000
↳ can be used for contribution this thing; as easy; not much changed.

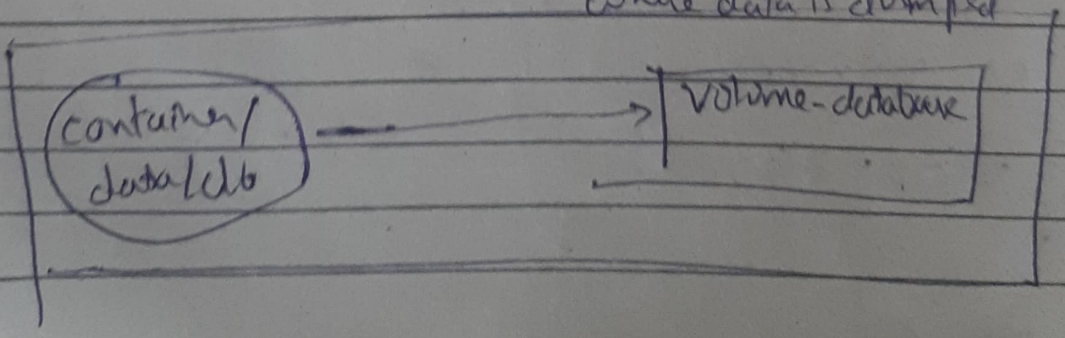
Volumes and Networks

↳ Run DBs/Redis/Aux Services.
↳ easy cleanups

Volume → local DB to retain info across restart
Network → one docker container to commute with other

Volumes → persist data across start.
↳ Things like database

docker volume create volume-database
docker run -v volume-database:/data/db -p 27017:27017 mongo
(where data is dumped)



docker exec it [id] /bin/bash → puts inside the container

docker run -v volume3:/data/db -p 27017:27017 mongo

↳ let persist data somewhere till the time you have volume here you can keep mounting your containers to like folder

docker kill <container_id>
docker volume rm volume-database

Backend server → go up; down

DB → persistent; permanent

To sort of cache the entry at website →

```
Const express = require('express');
Const mongoose = require('mongoose');

Const app = express();
Const port = 3000;

mongoose.connect("mongodb://localhost:27017/mydatabase", {});

Const EntrySchema = new mongoose.Schema({
    text: String,
    date: { type: Date, default: Date.now() }, });

Const Entry = mongoose.model('Entry', EntrySchema);
```

```
app.get('/', async (req, res) => {
    try {
        const entry = new Entry({ text: 'This is an entry' });
        await entry.save();
        res.send('Entry added!');
    } catch (err) {
        res.status(500).send('Error occurred');
    }
});

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
});
```

Backend code on machine; DB on container
↳ native hence can connect

Containers to containers ∝ each has own network

ip address of domain name → ping [site name]

: Say → ping : google.com
name of domain

ping localhost

Container localhost:27017
↓
own network

↑ is not parent machine

One container can't talk to another

Network → Used to let talk one containers to another

docker network create my_custom_network

docker run -p 3000:3000 --name backend --network my_custom_network <image_tag>

docker run -v volume_database:/database
--name mongo
--network my-custom_network
-p 27017:27017
mongo

docker network create networks
docker network ls

docker run -p 3000:3000 --network networks --name backend to test4

↳ container started; attached to network S name is backend to

1:10:45

ENV variable

Docker_2

Hkirat

1:10:45

ENV Variable

Problem
- difficult to keep editing files
- not check in database URL
- language in dependent construct to send secret to process

```
mongoose.connect(process.env.MONGO_URI,{});
```

MONGO_URI = localhost node index.js

↳ pass a variable inside a JS process

Soln → pass them as env variable

docker run -p 3000:3000 -e MONGO_URI=mongo <image_tag>

build the img → run with right env variable

Multi Stage Builds

nodemon ⟶ npx nodemon index.js instead of node index.js

↳ starts on port 3000

enter file edit it & save it. → the server restarts

npm run dev ⟶ server restart automatically

go to script ⟶ vi package.json

```
"scripts": {
    "start": "node index.js",      → on production
    "dev": "nodemon index.js"      → local
}
```

↳ we will not build another docker file

↳ in docker file :- Multi Stage build

```
FROM node:20 AS base
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
```

image 1 name = "base"

base image

(1)

```
FROM base AS development
COPY . .
CMD ["npm", "run", "dev"]
```

image 2 name = "development"

dev image

derived (base image = 1)

```
FROM base AS production
COPY . .
RUN npm prune --production
CMD ["npm", "run", "start"]
```

production image

derived (base image = 1)

docker build --target development -t app:dev

↳ build development image

build production target

↑

6 step image

docker build . --target production -t app:prod

↳ name of img

docker run app:prod

↳ start production

Problem → lot of aux service to be used

mount to ~~working~~ dir

docker run -v .:/usr/src/app app:dev .

↳ edit js file

the container restarts

kill it ←

Change made in container gets propogated to local machine

etc ":" and ":/usr/src/app" are same thing

docker build . --target development -t app:dev

docker run -v .:/usr/src/app app:dev

↳ w/o any dependency you can work

Docker Compose.

↳ compose bunch of img together

Prob

lot of aux service to be used

Soln

run in detach mode

(load soln)

Good sol

Docker compose → define, run multi-container docker app

→ YAML file used to configure services

↳ single command to create, start services from configuration

Yaml file →

```
services:
  mongodb:
    image: mongo:latest
    ports:
      - "27017:27017"

  mysql:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
    ports:
      - "3306:3306"

  custom-app:
    build: ./
    ports:
      - "3000:3000"
```

taken as input

docker-compose up

all containers are connected via "network"

add volume too