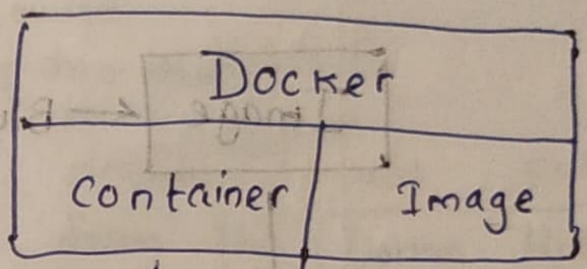
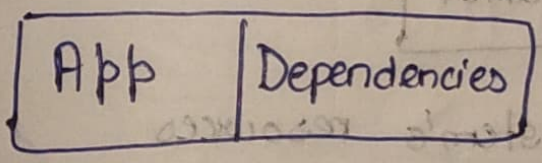
DOCKER

"It works on my machine"

→ A platform that helps us build containers.

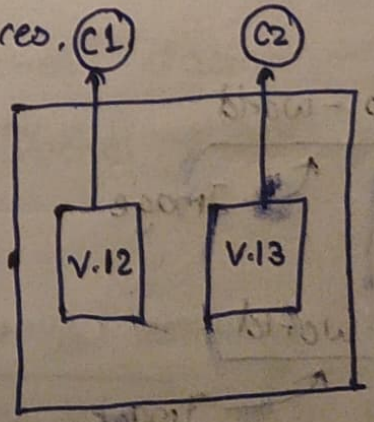




"A standard way of sharing application with its dependencies across a large team"

- ✓ Any OS
- ✓ Portable
- ✓ Light-weight

→ Container manages a separate environment for each instances.



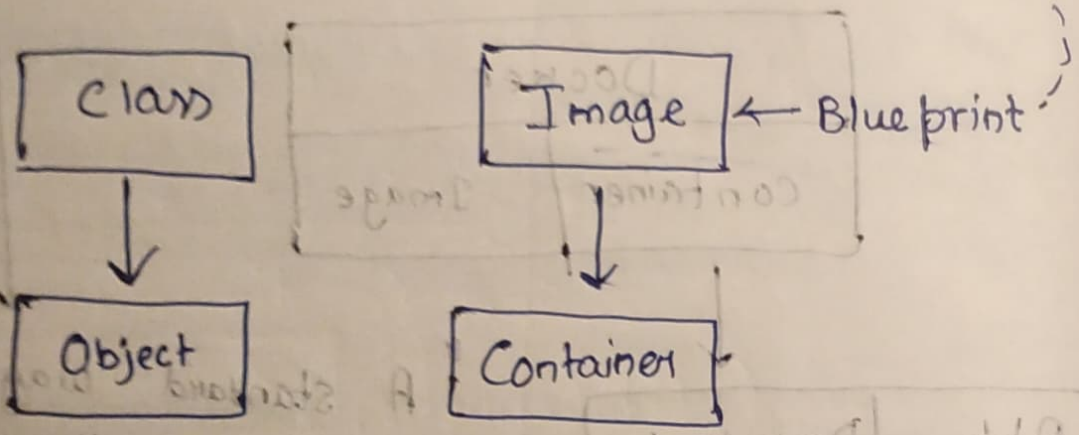
Docker Image -

→ Image ✗

→ Executable file ✓

→ Contains instructions to build container

→



→ Image doesn't use system's resources

→ Static snapshot / screenshot

Commands -

> docker -v / docker --version

> docker

> docker pull hello-world

Image

> docker images

> docker run hello-world

Image

Steps -

1) The docker client (CLI) contacted the docker daemon.
   - Sent the command (docker run hello-world)
   - Background service (core of docker desktop)

2) The docker daemon pulled the "hello-world" image from the Docker Hub.
   - image on (amd 64)
   - A public image repository

3) The docker daemon created a new container from that image which runs the executable that produced the output you are currently reading.

4) The docker daemon streamed that output to the Docker client, which sent it to your terminal.

Output is comming ---

From container → to daemon → to client → to terminal

Commands –

> docker pull "image-name"

↖ only pulls image from docker hub.

> docker run "image-name"

↖ pulls image from docker hub if not locally available then create container.

> docker run -it "image-name"

↖ (Interactive)/(tty)

↓ i

Keeps STDIN open, so you can type I/P"

↓ t

"Allocates a terminal for you inside the container"

it → "Let me interact with the container via terminal."

> exit

↖ exit the running container (with its own terminal)

> docker ps -a

↖ To check all container (running/closed)

> docker ps

↖ To check all running containers

> docker start / stop "container_name" or "container_id"

↖ To start / stop already existing container.

> docker rmi "image-name" or - id ← Remove image

> docker rm "container-name" or - id ← Remove container

> docker pull "image-name" : version

↖ pull image with specific version

> docker run -d "image-name"

↳ detach

→ By default all containers runs on attached mode

Example –

docker run -d -e MYSQL_ROOT_PASSWORD = secret mysql

-d ↙ detach mode

-e ↳ environment variable

mysql ↳ name

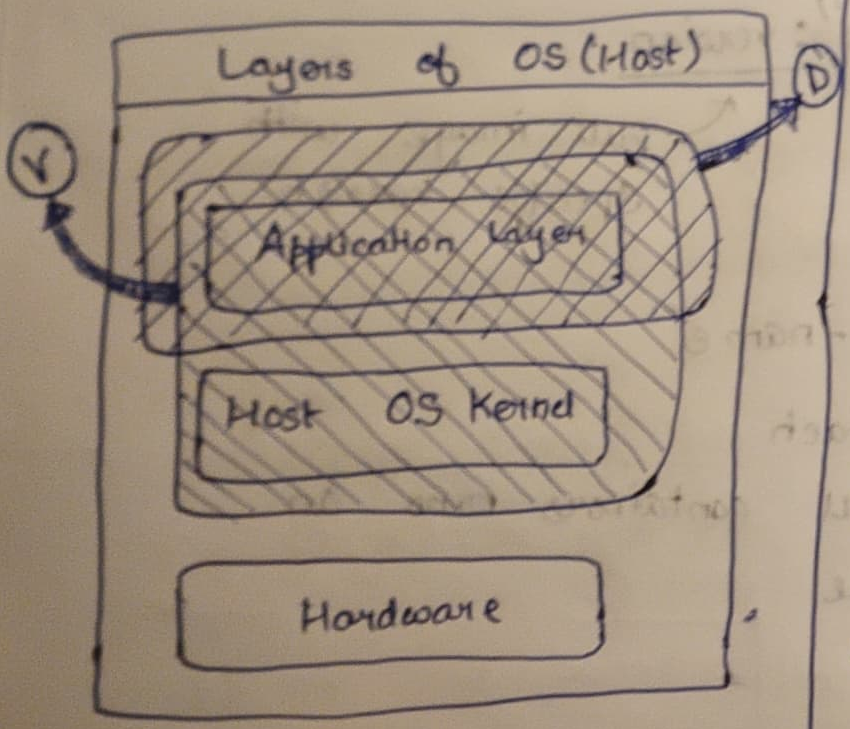
> docker run -d -e MYSQL_ROOT_PASSWORD = secret --name mysql-older mysql:8.0

--name ↳ new name

mysql:8.0 ↳ image

→ Difference between virtual machines and docker container.

Virtual Machines	Docker Container
→ Size : Large	: Small
→ Compatability : Every OS	: Originally for Linux, but we are able to run these Linux commands is only due to presence of docker desktop. ↓ Adds a lightweight hypervisor layer, which internally uses a light weight Linux distribution.
→ Architecture : Hypervisor-based	: Container-based
→ OS layer : Each VMs has its own OS	: Shares host OS kernel
→ Isolation : Full OS-level isolation	: Process-level isolation



→ Docker uses host OS's kernel, it virtualise application layer of host.

→ VM virtualise both application layer & host OS's kernel but uses its own kernel

virtualization: Application Layer + Host OS Kernel	: Application layer, this virtualization of only application layer results in lightweight.
Kernel use: Own	: Host OS's
Boot time: Minutes	: Seconds.
Portability: ~~High~~ Less	: ~~less~~ High
Deployment Speed: ~~Fast / Scalable~~ Slower / more complex	: Fast / Scalable
Security: High, strong isolation (own kernel)	: Low, shared kernel (with OS)

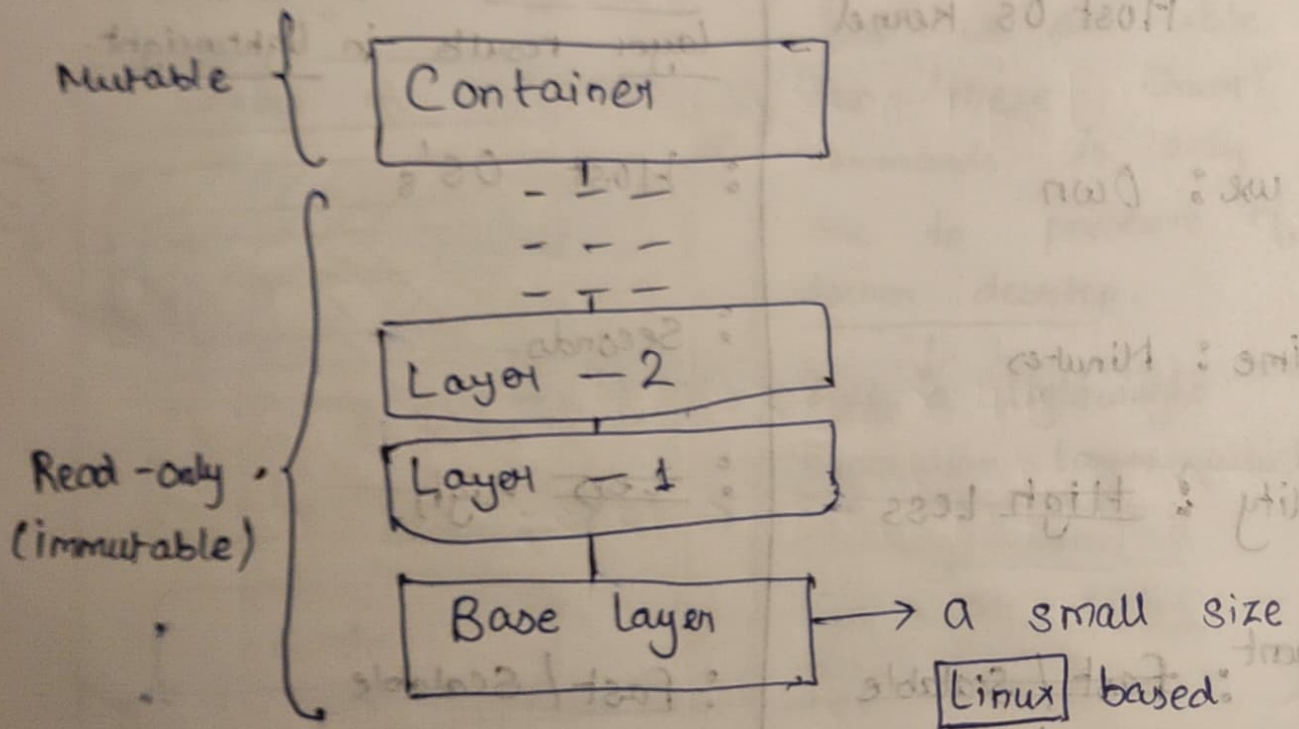
↳ ps -a / ps

Container

↳ Stop → remove → Image ← images

↳ remove

⇒ Docker image layers —



Example –

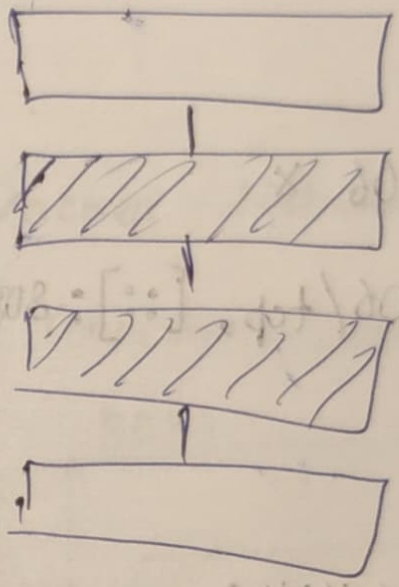
Image – X

Version – X : latest , X : 4.0

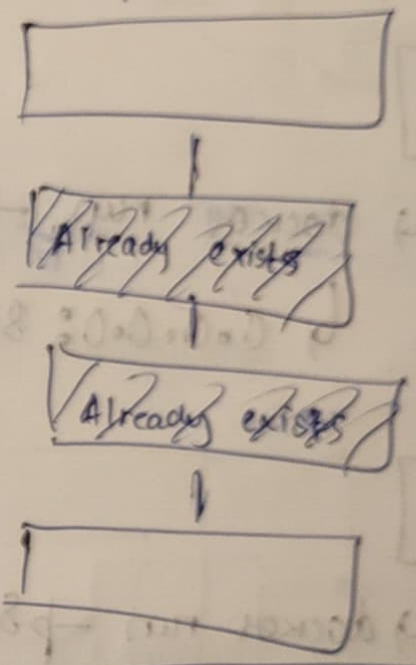
image → x : latest
(Already set-up)

image → x : 4.0
(Setting up image - with new version or different version)

» docker pull x:latest



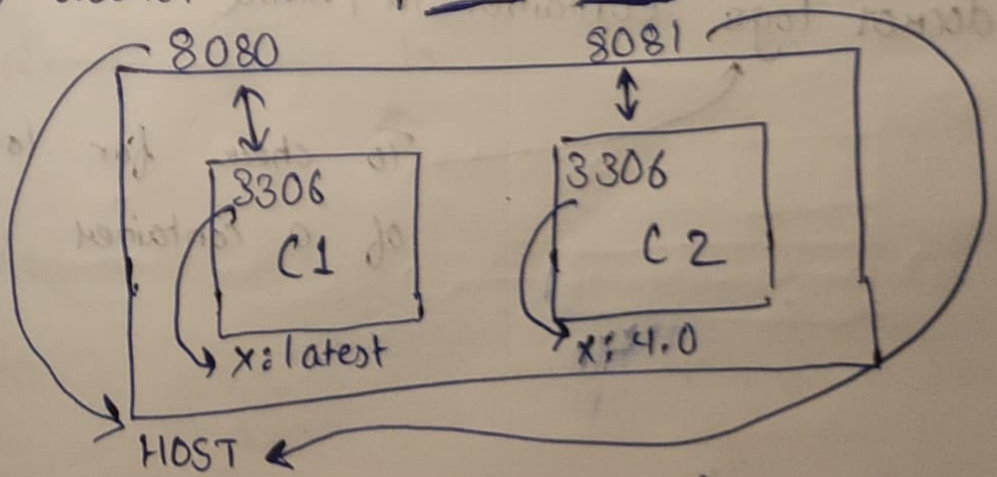
» docker pull x:4.0



⇒ Port Binding

→ Binding port of HOST machine with port of docker container.

→ docker run -p8080:3306 IMAGE_NAME



host port → map → container port

→ -p hostport : containerport

C1

↳ docker run -p 8080:3306 x

↳ 0.0.0.0:8080 → 3306/tcp, [::]:8080 → 3306/tcp

C2

↳ docker run -p 8081:3306 x:4.0

↳ 0.0.0.0:8081 → 3306/tcp, [::]:8081 → 3306/tcp

⇒ Troubleshoot Commands

> docker logs container_id / name

To check for logs of a container

> docker exec -it cont_ID /bin/bash

To access bash

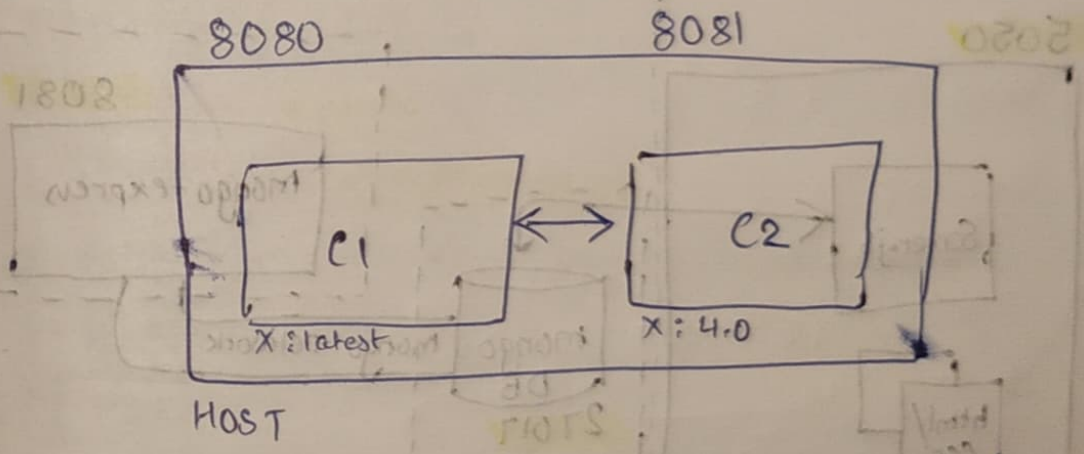
exec → To run additional commands on an already running container

> docker exec -it cont_ID /bin/sh

To access shell

Docker Network

↳ Connection between two containers



→ No need of any port or local host

→ Create new docker network (isolated)

> docker networ ls

↖ To get all networks

> docker network create mongo-network

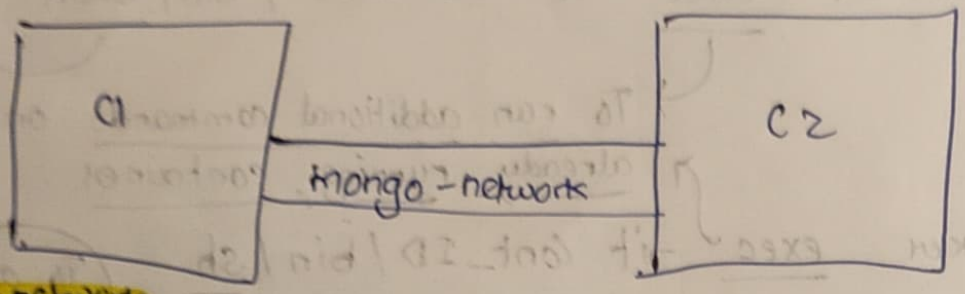
↳ It will have a driver → Bridge

> docke network rm mongo-network

Types
- default ← Bridge
- we have created in test ← custom

② host network

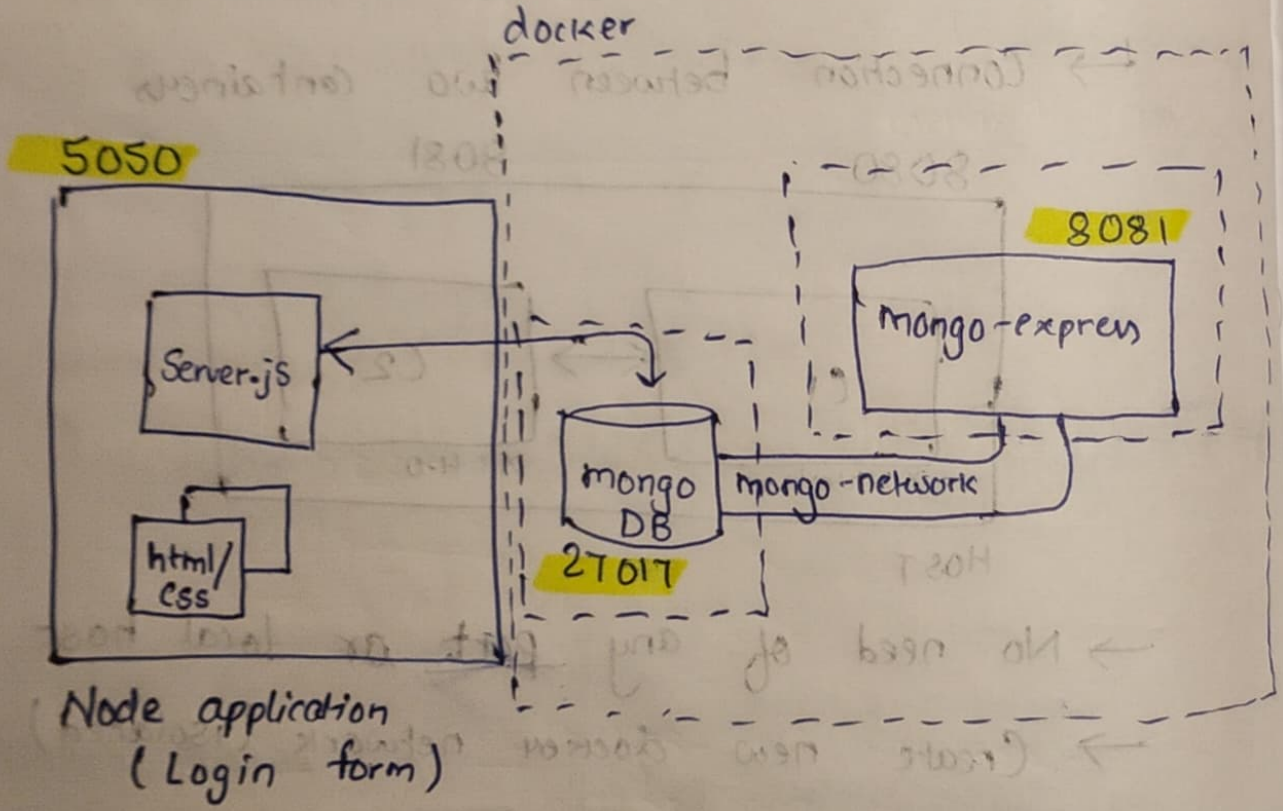
↳ container with no ip address / uses host network



③ null network

↳ container with no connection with host ~~def~~ device or any other container.

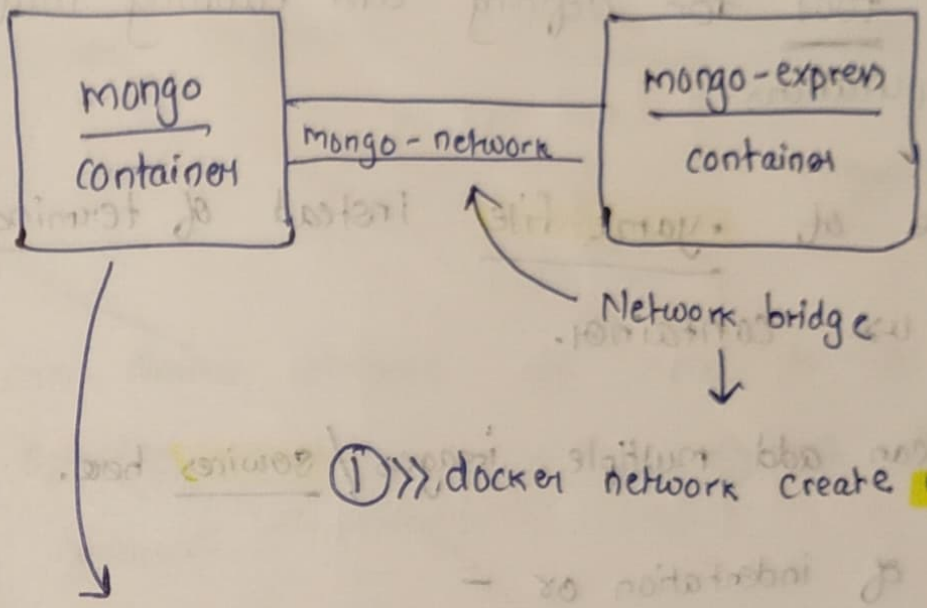
➡ Developing an application with Docker



Images –

1) mongo ⟶ Database

2) mongo-express ⟶ UI for mongo db.



① >> docker network create mongo-network

② >> docker run -d \
> -p27017:27017 \
> -- name mongo \
> -- network mongo-network \
> -e MONGO_INITDB_ROOT_USERNAME=admin \
> -e MONGO_INITDB_ROOT_PASSWORD=1234 \
> mongo

③ >> docker run -d \
> -p8081:8081 \
> -- name mongo-express \
> -- network mongo-network \
> -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin \
> " " _ADMINPASSWORD=1234 \
> " " _URL="mongodb://admin:1234@mongo:27017" \

> mongo-express

⇒ Docker compose –

→ A tool for defining and running multi-container applications.

→ Use of .yaml file instead of terminal to use container.

→ We can add multiple images / sewices here.

→ Use of indentation or –

Example –

mongodb.yaml

```
version: "3.8"
sewices:
  mongo:
    image: mongo
    ports:
    - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: 1234
```

→ It contains instructions for .yaml file that which container it will create and use.

Start/create

>> docker compose -f fileName.yaml up -d

>> docker compose -f fileName.yaml down

Delete/ remove

→ To run docker compose we need to move to the specific directory where .yaml files are present.

→ No explicit network defined : Automatically default network are created by docker compose.

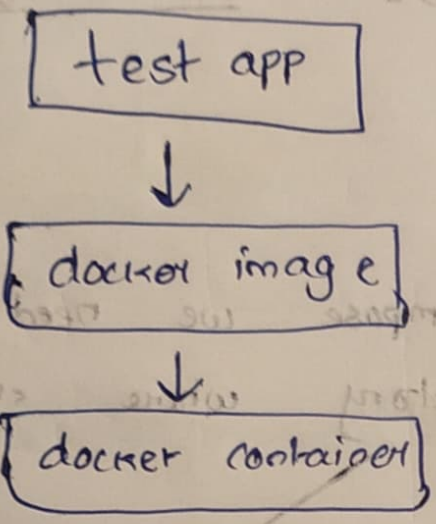
Output

✓	Network	docker-testapp-main_default	Created
✓	Container	docker-testapp-main-mongo-expres-1	Started
✓	Container	docker-testapp-main-mongo-1	started

Output

✓	Container	testapp_main^mongo-expres-1	Removed
✓	Container	testapp-main-mongo-1	Removed
✓	Network	testapp-main-default	Removed

⇒ Dockerizing our App -



↳ The process of converting an application into a running docker container is called dockerizing our application.

→ Right now, I will do it by own but in CICD pipeline we have a tool named Jenkins.

→ Docker file – (Blueprint)

– Contains instructions for dockerizing application

Dockerfile :

FROM ⟶ base image

WORKDIR ⟶ setting working directory

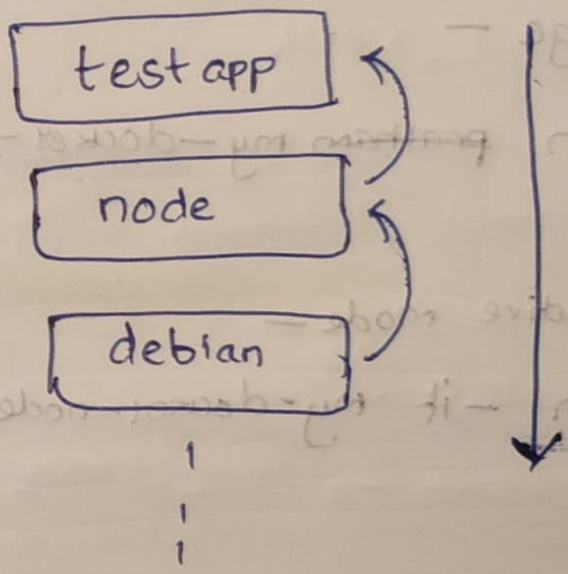
COPY ⟶ (host) ⟶ (image) Copy files from host to docker image

RUN ⟶ To run commands (It can be multiple)

CMD ⟶ The last command (only one) ⟶ node server.js

EXPOSE ⟶ To expose app port

ENV ⟶ environment variable



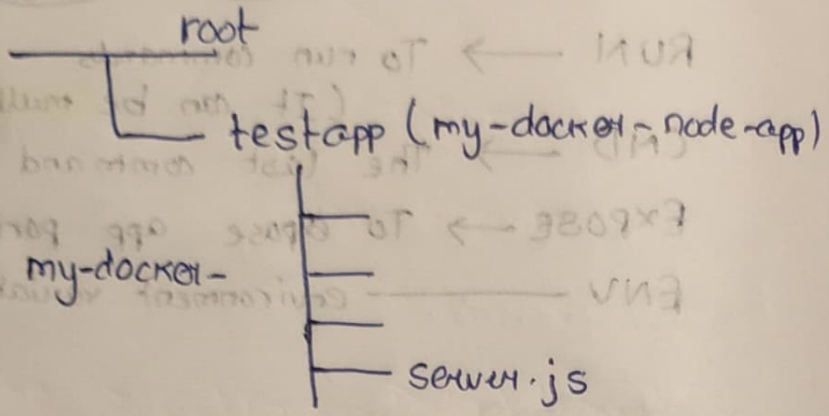
For example —

Dockerfile

```
FROM node
ENV MONGO_DB_USERNAME=admin MONGO_DB_
    PASSWORD=1234
RUN mkdir -p my-docker-node-app
```

```
COPY . /my-docker-node-app
CMD ["node", "/my-docker-node-app/server.js"]
```

To build this Dockerfile — (image)



>> docker build -t my-docker-node-app .

To run this image —

>> docker run ~~prathan~~ my-docker-node-app

To run in interactive mode —

>> docker run -it my-docker-node-app bash

⟹ Publish Images —

>> hub.docker.com
>> repositories — create one
>> docker build -t prathamraj/my-docker-node-app .
>> docker login in terminal
>> docker login

>> docker push prathamraj/my-docker-app
(Now available on docker hub)

>> docker pull " "

⇒ Docker Volumes —

↳ To store data (persistent data) for containers.

→ Persist data even we stop or delete the container.

For mongodb.yaml —

↳ volumes:
— /absolute_host_path : /absolute_container-path

Example -

volumes:
- /Users/rjpratham/OneDrive/Desktop/Docker/Data : /data/DB

>> docker compose -f mongodb.yaml up

>> " " down

For terminal —

↳ >> docker run -it -v /Users/rjpratham/OneDrive/Desktop/Docker/Data:/data/DB ubuntu

>> docker volume ls

>> docker volume create VOL_NAME

Isolated & not attached to any container.

To create custom volumes (also known as named volumes)

By default its location on windows will be → C:\ProgramData\docker\volumes

volume

>> docker rm VOL_NAME

→ To attach named volumes with running containers –

I) Named volumes **

docker create/manage volume

>> docker run -v VOL-NAME : CONT_DIR

II) Anonymous Volumes

>> docker run -v MOUNT_PATH

↖ No path/name is known for volume

III) Bind Mount

>> docker run -v HOST_DIR : CONT_DIR

Volume Managed by host OS here

→ flags –

* --volumes
  -v

* --mount

>> docker volume prune

→ Delete unused volumes locally

→ By default it targets anonymous volumes