DOCKER

- **Reference video:** Youtube
- **What is docker?**
  - Docker uses the OS kernel of the host machine where the docker container is running eg: our server
  - Docker virtualizes the applications not the OS
  - If the docker image is not compatible with our local machine OS versions then use docker toolbox to settle up the conflicts of the OS.
- **What is the container?**
  - layers of images
  - first layer will be of the linux based image for OS environment
  - One container may need more than one images download each image separately
  - For egdata_base image, etc.
  - Images are available on docker hub for free
  - Container is the running state of the images.
  - The upper most image will be our own application running in the container.
  - container gives the environment to the image to actually run.
- **Difference between VM and container?**
  - VM virtualizes both OS and application, but container virtualizes the app running on it and container will use the OS enviorment of the host_machine itself(host machine can also be one VM running on our laptop or running on the AWS cloud i.e EC2 instance).
- **Advantage of the containers:**
  - Let's say our application uses 2 diff types of mySQL versions. Then I can make one container running MYSQL V1 and another container running MYSQL V2.
- **What is an image?**
  - Image is the actual package that contains all the dependencies, environment, configurations, application code needed to run the image.
- **Difference between container and image**
  - Container is the running environment for image
  - Means container will give the environment, file system needed to store some data, etc to the image(our application or predefined some image downloaded from the docker hub).
  - Container will bind one port to the image running on the container.
- **What do we mean by port mapping of containers?**
  - On 1 host machine/ VM more than 1 docker container can run parallely.
  - 2 containers can run on the same port in the same host machine.
  - Now question arises that how will the API call bifurcated between the 2 containers(basically 2 images running on different containers).
  - Now let's say there are 2 containers running on the same port 3000.
  - Here the concept of binding the host port to the container port comes in.

- ○ Whenever a container starts running on a VM/host machine, the machine will assign one port to that container(eg: 5000). Now, this 5000 port will be mapped with one of the ports of the container(eg:3000).
- ○ Now for the other container having the same port(eg: 3000) will be binded to another port number assigned by the machine(eg: 6000).
- ○ So, whenever the request comes it will come on the port of the host machine/ VM not directly to the container.
- ○ According to the mapping done between the machine port and container port(basically image is being binded to the host port) the request will be forwarded to that specific container(image-our sample app) by the host machine/ VM.
- ○ It means the API calls will be made on the port number of the host machine/ VM and not the actual container.

- **What is stored in the docker container?**
    - Docker uses storage drivers to store image layers, and to store data in the writable layer of a container.
    - The container's writable layer does not persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime.
- **General Points**
    - Whenever we are deploying the app the database used by that app will run on different containers.
    - It simply means that let's say we have one java-script app and that app uses mongodb as the data-base for the storage.
    - We will use the image from the docker hub of the mongo db directly. For our app code one separate custom docker image will be formed which will be stored into some private docker repository of the organisation.
    - Now when we want to run both the images, server/host_machine will pull both the images mongo-db from the public docker hub and app-code image from the private docker repository and each image will be deployed on individual separate containers running parallely.
    - There will be some intermediator called jenkins which will make the docker image of our app-code based on the given artifacts of the app-code. (artifacts like the environment info, version info of the language used by the application, etc)

DOCKER

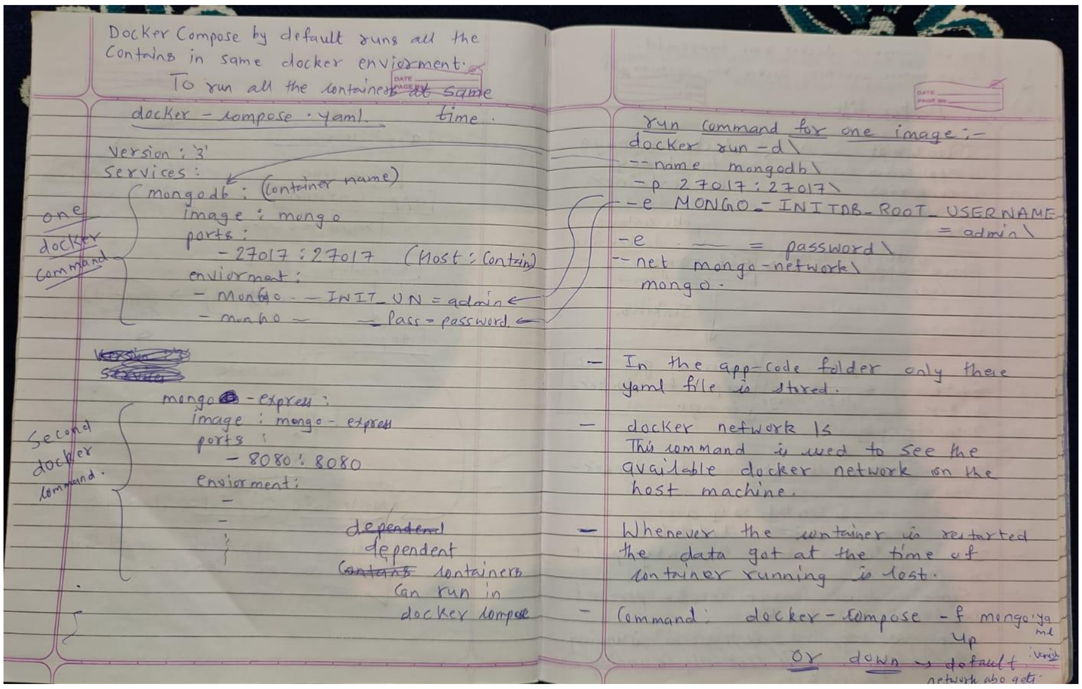
DOCKER COMMANDS TABLE

When to use	Command
build image	docker build -t <image_name>:<tag> <path_to_Dockerfile>
run image	docker run -p <host_port>:<app_port> <image_name>:<tag>
list running containers	docker ps
list all containers	docker ps -a
start container	docker start <container_name/id>
stop container	docker stop <container_name/id>
remove container	docker rm <container_name/id>
remove image	docker rmi <image_name>:<tag>
remove all containers	docker rm -f $(docker ps -aq)
remove all images	docker rmi -f $(docker images -aq)
get container id	docker container ls –quiet –filter name=^<container_name>$
get all info of container	docker inspect <container_name/id>
check container status	docker inspect -f '{{.State.Status}}' <container_name/id>
live logs of container	docker logs <container_name/id> -f
go inside file-system of container	docker exec -it <container_name/id> /bin/sh

Reference: Docker commands official

DOCKER

Additional Images



Docker Compose by default runs all the Contains in same docker enviorment.

To run all the containers ~~at~~ same time.

docker - compose . yaml.

```
Version : '3'
services :
  mongodb : (container name)
     image : mongo
     ports :
       - 27017 : 27017   (Host : Contain)
     enviorment :
      - Mongo - - INIT_UN = admin
      - mongo -        - Pass = password.
```

one docker command

~~Version '3'~~
~~Service~~

```
  mongo - express :
     image : mongo - express
     ports :
       - 8080 : 8080
     enviorment :
       -
       -
```

Second docker command.

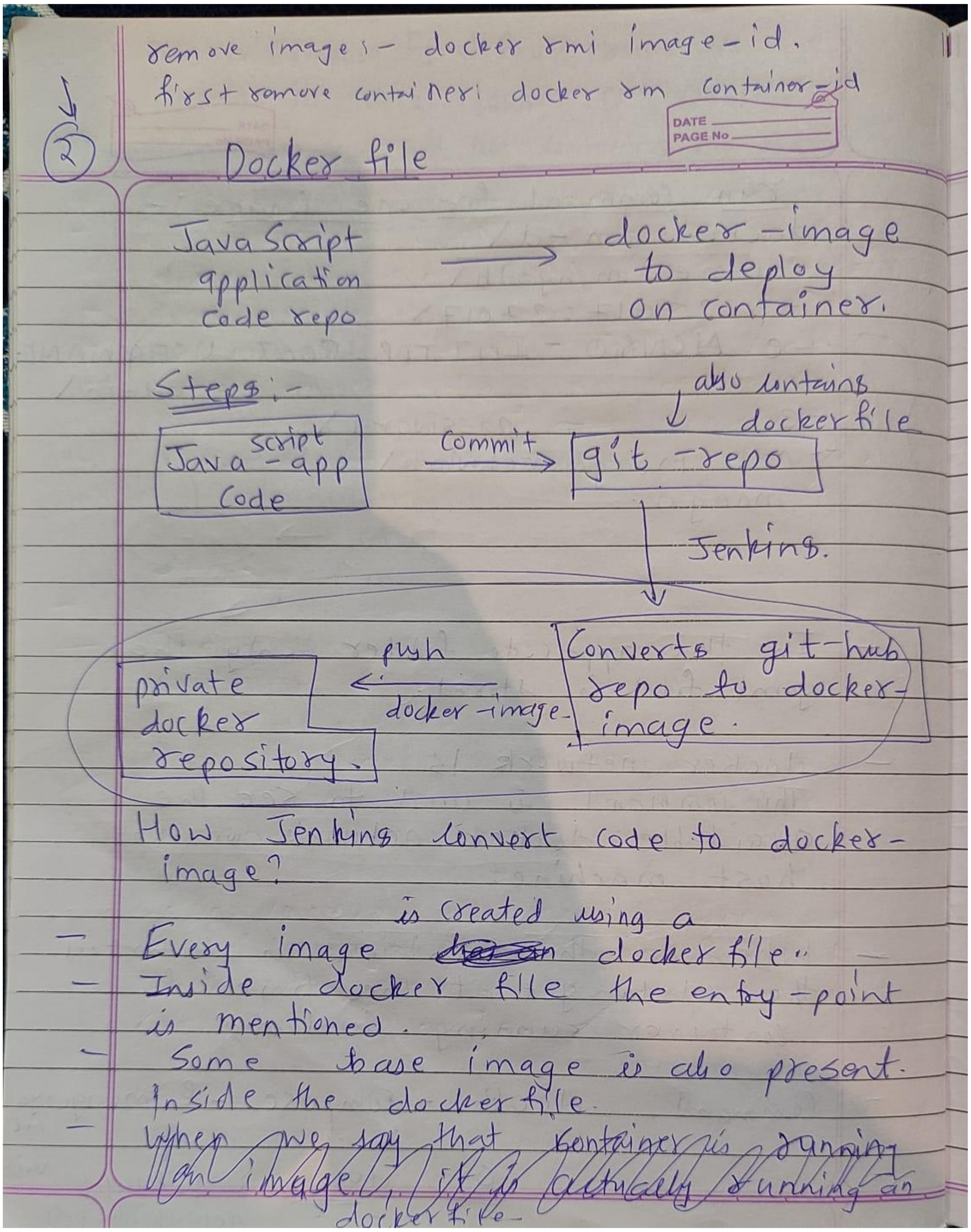
~~dependend~~ dependent ~~Contans~~ containers can run in docker compose

run command for one image :-

```
docker run -d \
--name mongodb \
 -p 27017 : 27017 \
-e MONGO - INITDB_ROOT_USERNAME
                          = admin \
-e  ___  = password \
--net mongo-network \
 mongo.
```

- In the app-code folder only there yaml file is stored.
- docker network ls
  This command is used to see the available docker network on the host machine.
- Whenever the container is restarted the data got at the time of container running is lost.
- Command : docker - compose - f mongo.yaml up
  or down → default network also gets vanish

DOCKER



remove images :- docker rmi image-id.
first remove containers: docker rm container-id

DATE
PAGE No

2

Docker file

$$\begin{tikzcd} \text{JavaScript application code repo} \arrow[r] & \text{docker-image to deploy on container.} \end{tikzcd}$$

Steps :-

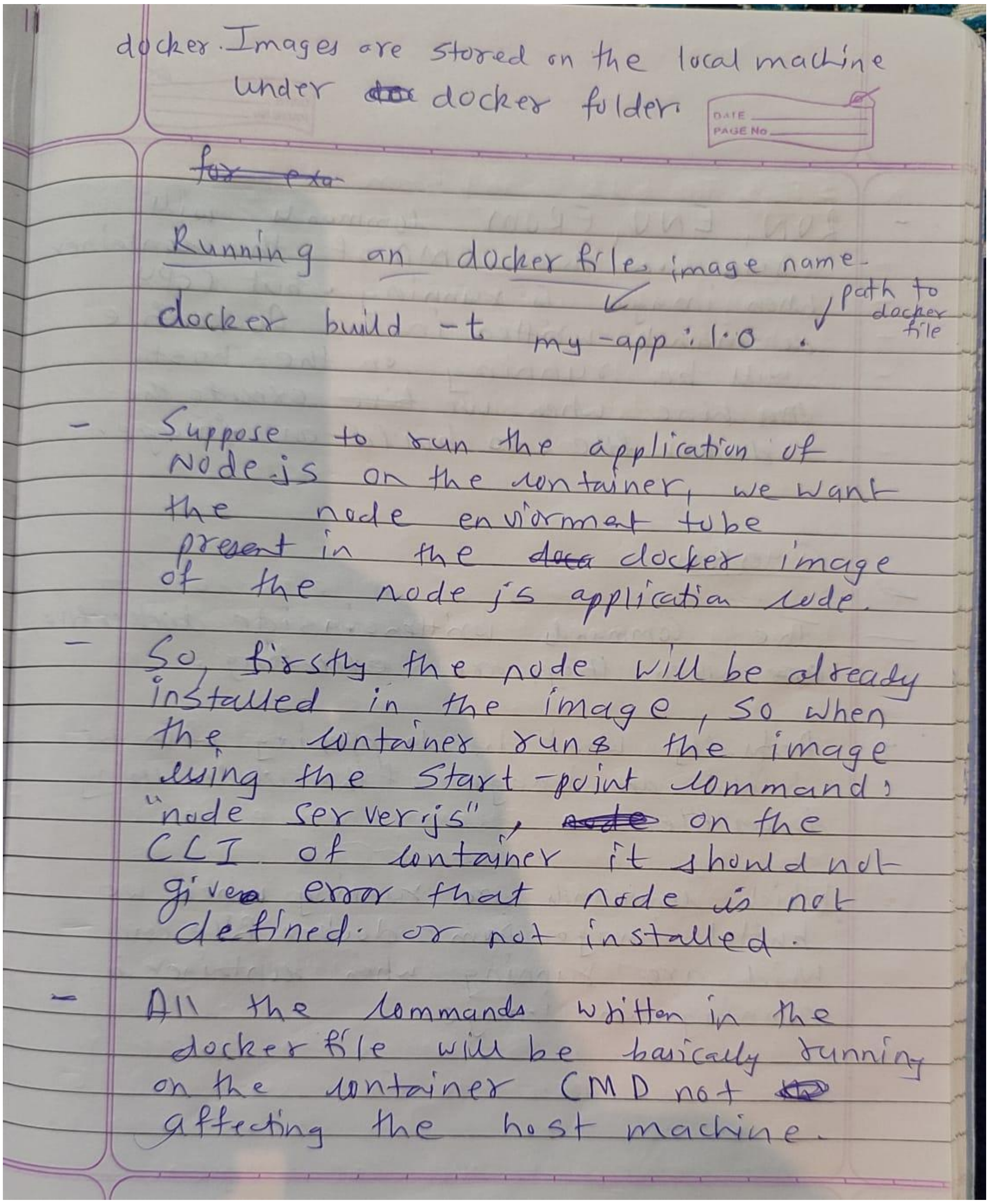
also contains docker file

$$\begin{tikzcd} \boxed{\text{Java script-app Code}} \arrow[r, "\text{commit}"] & \boxed{\text{git-repo}} \arrow[d, "\text{Jenkins.}"] \\ \boxed{\text{private docker repository.}} & \boxed{\text{Converts git-hub repo to docker-image.}} \arrow[l, "\text{push}", "\text{docker-image}"'] \end{tikzcd}$$

How Jenkins convert code to docker-image?

- Every image ~~has an~~ is created using a docker file.
- Inside docker file the entry-point is mentioned.
- Some base image is also present. Inside the docker file.
- ~~When we say that container is running an image, it is actually running an~~ docker file.

DOCKER



docker. Images are stored on the local machine under ~~doc~~ docker folder.

~~for exa~~

Running an docker file.

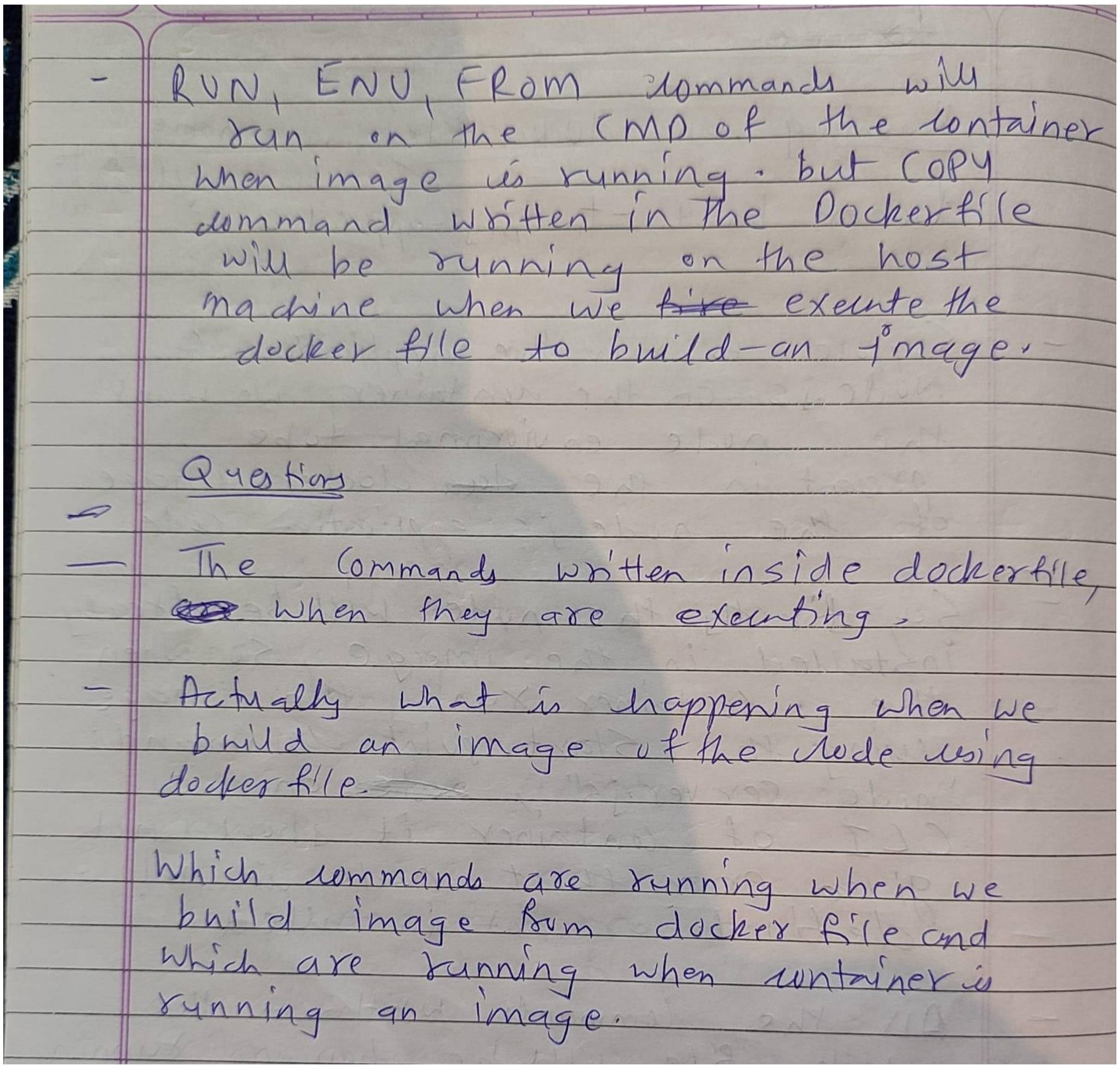
image name

path to docker file

```
docker build -t my-app:1.0 .
```

- Suppose to run the application of Node.js on the container, we want the node enviornmat tube present in the ~~doca~~ docker image of the node js application code.
- So, firstly the node will be already installed in the image, so when the container runs the image using the start-point command; "node server.js", ~~node~~ on the CLI of container it should not give error that node is not defined. or not installed.
- All the commands written in the docker file will be basically running on the container CMD not ~~[illegible]~~ affecting the host machine.

DOCKER



- RUN, ENV, FROM commands will run on the CMD of the container when image is running. but COPY command written in the Dockerfile will be running on the host machine when we ~~fire~~ exeute the docker file to build-an image.

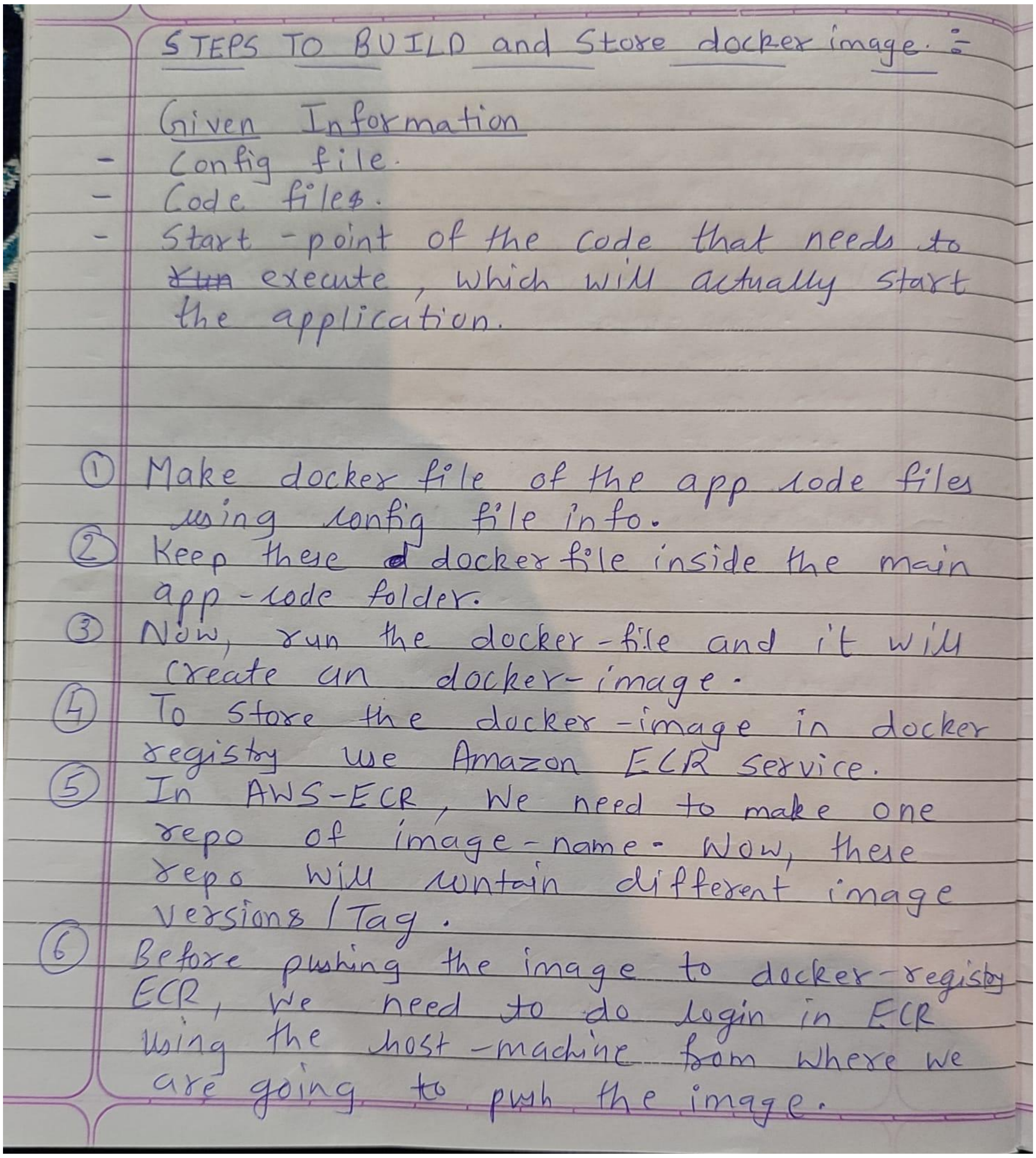
Questions

— The commands written inside dockerfile, when they are exeuting.

- Actually what is happening when we build an image of the code using docker file.

Which commands are running when we build image from docker file and Which are running when container is running an image.

DOCKER



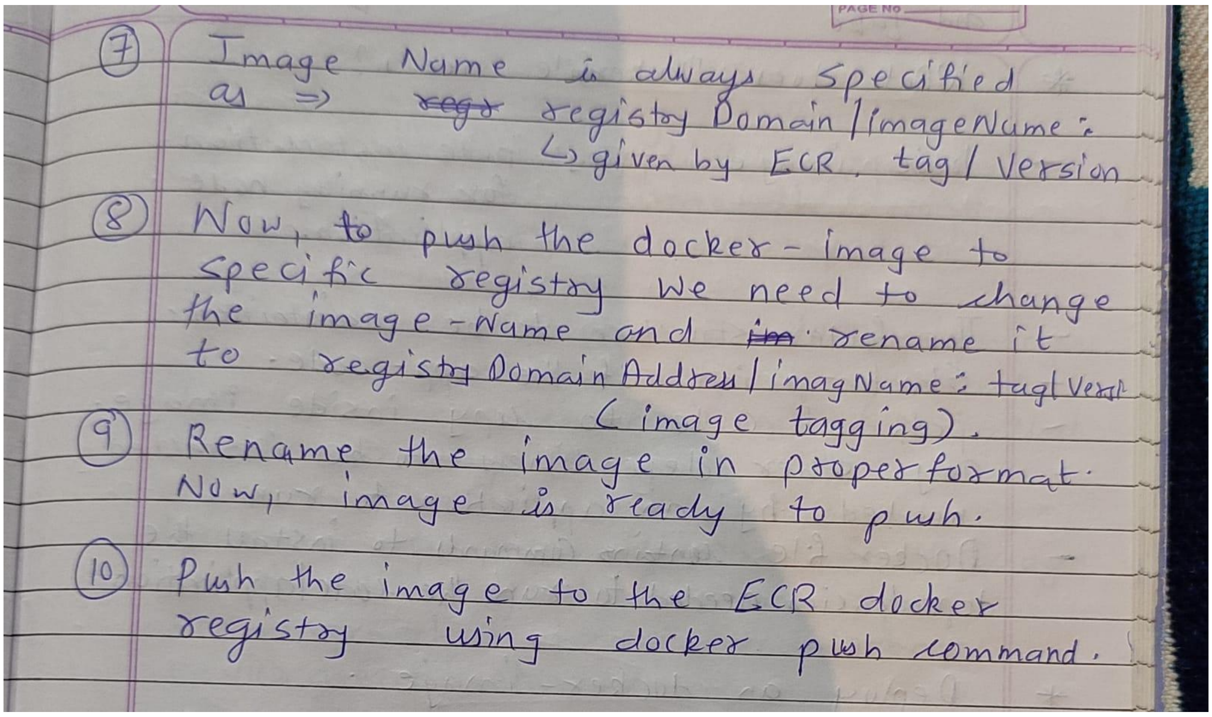
STEPS TO BUILD and Store docker image :-

Given Information

- Config file.
- Code files.
- Start-point of the code that needs to ~~run~~ execute, which will actually start the application.

1. Make docker file of the app code files using config file info.
2. Keep these ~~d~~ docker file inside the main app-code folder.
3. Now, run the docker-file and it will create an docker-image.
4. To store the docker-image in docker registry we use Amazon ECR service.
5. In AWS-ECR, we need to make one repo of image-name. Now, these repo will contain different image versions/Tag.
6. Before pushing the image to docker-registry ECR, we need to do login in ECR using the host-machine from where we are going to push the image.

DOCKER



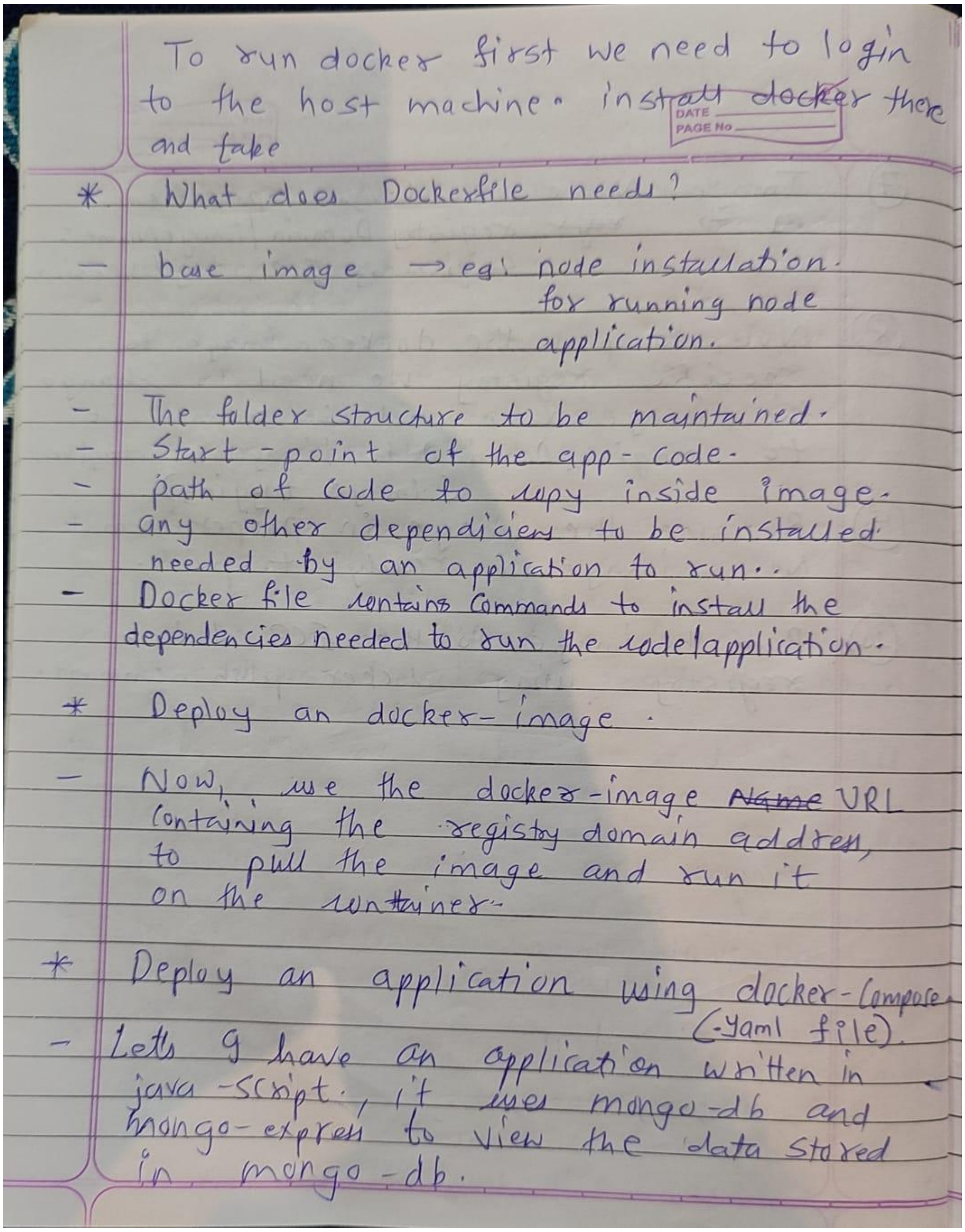
⑦ Image Name is always specified as ⇒ ~~regr~~ registry Domain/imageName : tag/Version
↳ given by ECR

⑧ Now, ~~to~~ push the docker-image to specific registry we need to change the image-Name and ~~im~~ rename it to registry Domain Address/imagName: tag/Versi
(image tagging).

⑨ Rename the image in proper format.
Now, image is ready to push.

⑩ Push the image to the ECR docker registry using docker push command.

DOCKER



To run docker first we need to login to the host machine. ~~install docker~~ there and take

* What does Dockerfile needs?

- base image → eg: node installation for running node application.

- The folder structure to be maintained.
- Start-point of the app-code.
- path of code to copy inside image.
- any other dependiciens to be installed needed by an application to run..
- Docker file contains Commands to install the dependencies needed to run the code/application.

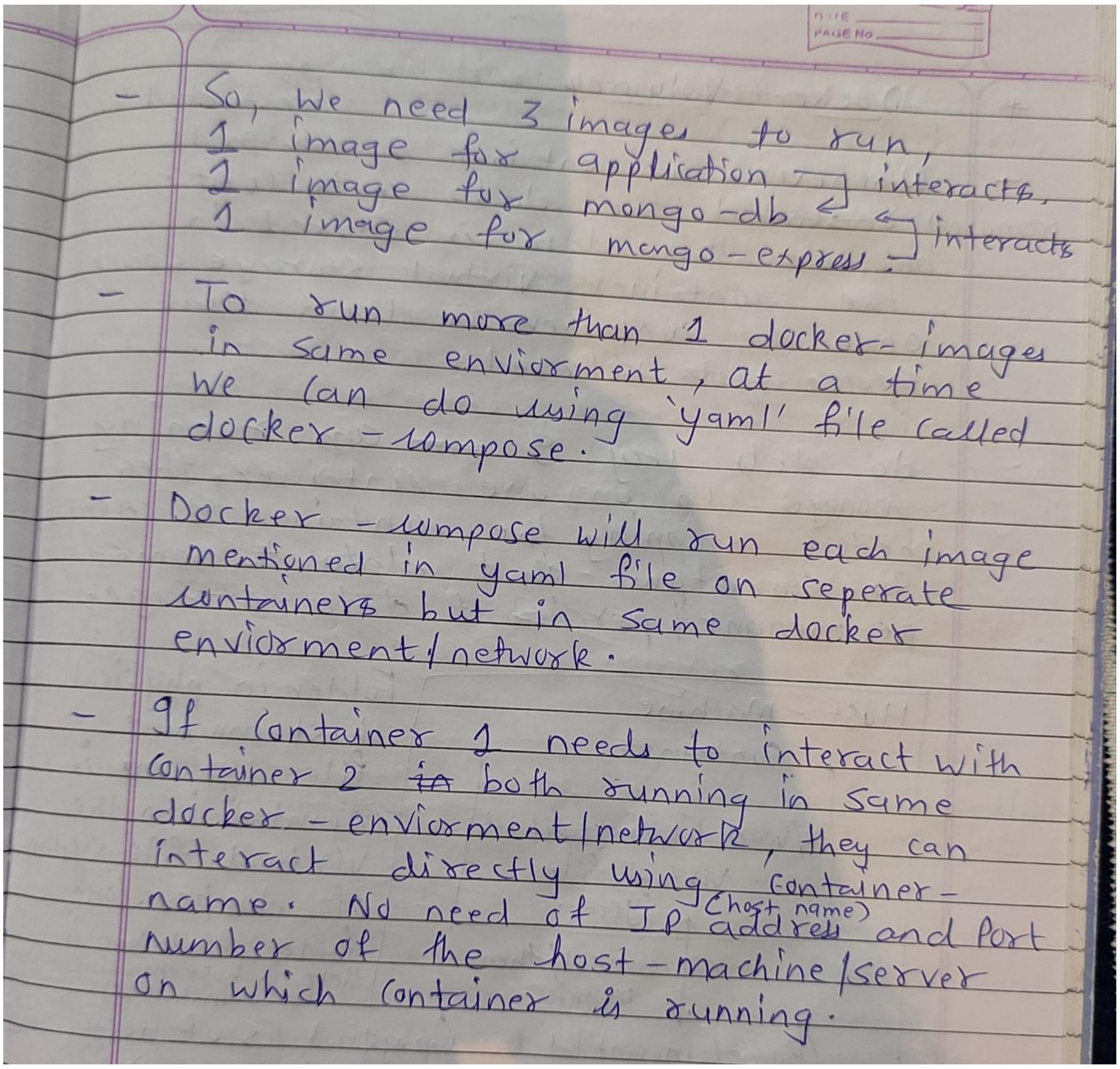
* Deploy an docker-image.

- Now, use the docker-image ~~Name~~ URL containing the registry domain address, to pull the image and run it on the container.

* Deploy an application using docker-Compose (.yaml file).

- Lets 9 have an application written in java-script, it uses mongo-db and mongo-express to view the data stored in mongo-db.

DOCKER

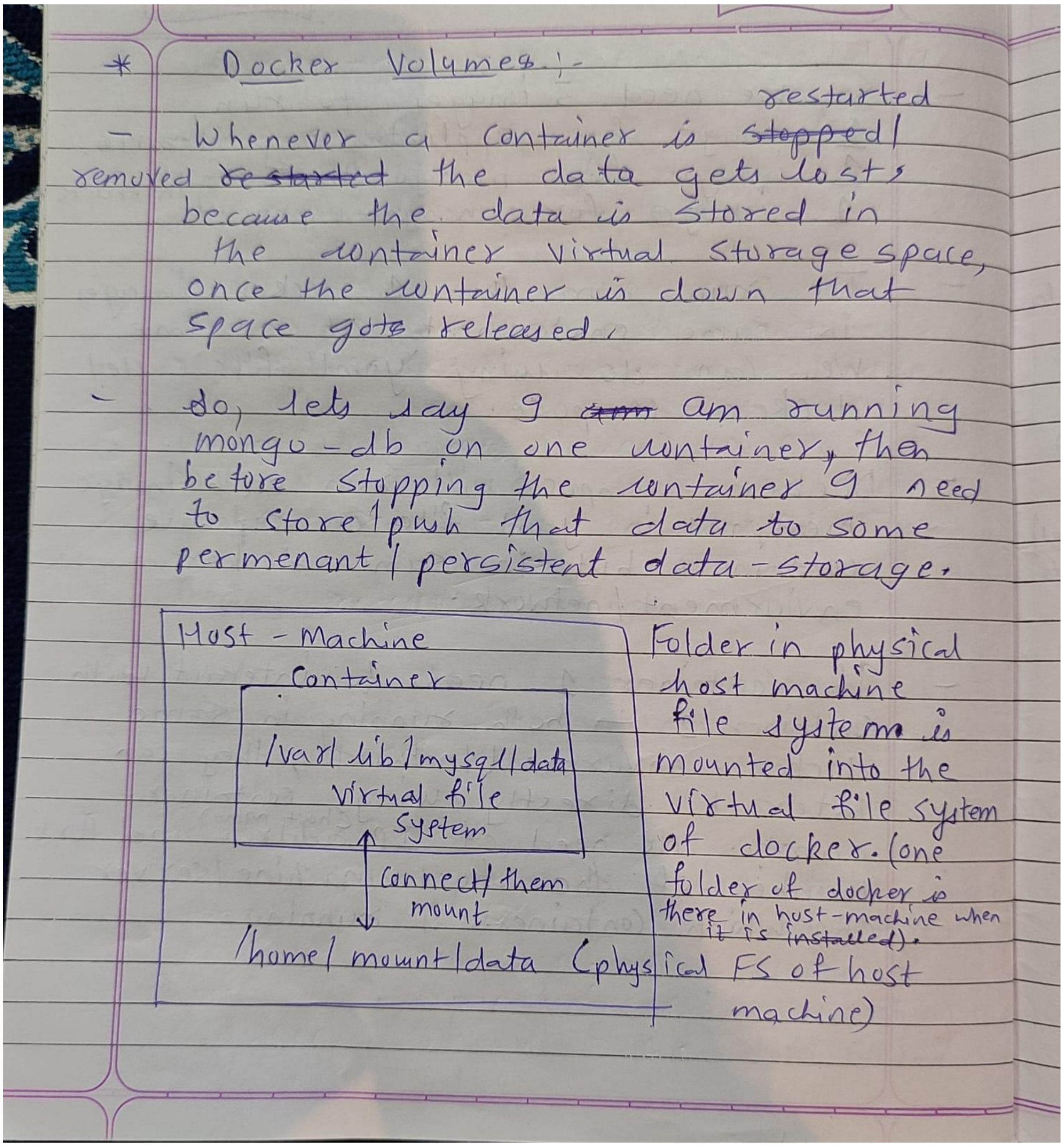


- So, we need 3 images to run,
  1 image for application ⏋ interacts,
  1 image for mongo-db ↲ ↵ ⏋ interacts
  1 image for mongo-express ⌟

- To run more than 1 docker-images in same enviorment, at a time we can do using 'yaml' file called docker-compose.

- Docker-compose will run each image mentioned in yaml file on seperate containers but in same docker enviorment/network.

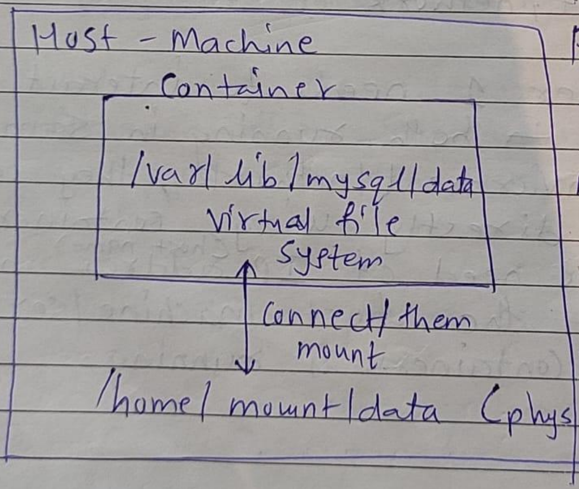
- If container 1 needs to interact with container 2 ~~in~~ both running in same docker-enviorment/network, they can interact directly using 'container-name'. (host name) No need of IP address and Port number of the host-machine/server on which container is running.



* Docker Volumes :-

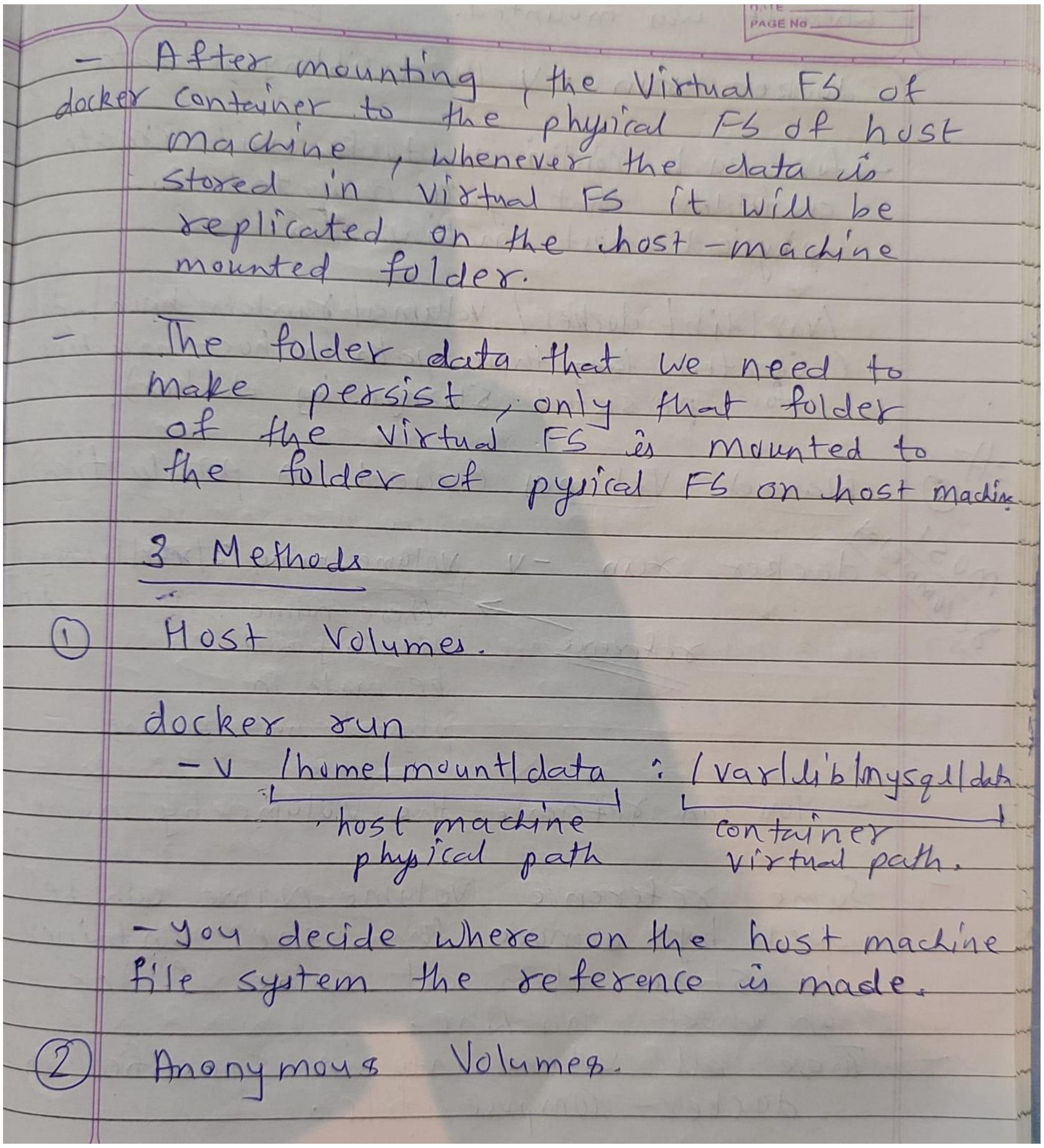
- Whenever a container is ~~stopped~~ restarted / removed ~~restarted~~ the data gets lost, because the data is stored in the container virtual storage space, once the container is down that space gets released.

- So, lets say I ~~am~~ am running mongo-db on one container, then before stopping the container I need to store/push that data to some permenant/persistent data-storage.



Folder in physical host machine file system is mounted into the virtual file system of docker. (one folder of docker is there in host-machine when it is installed).

DOCKER



- After mounting, the Virtual FS of docker container to the physical FS of host machine, whenever the data is stored in virtual FS it will be replicated on the host-machine mounted folder.

- The folder data that we need to make persist, only that folder of the virtual FS is mounted to the folder of pysical FS on host machine.

3 Methods

① Host Volumes.

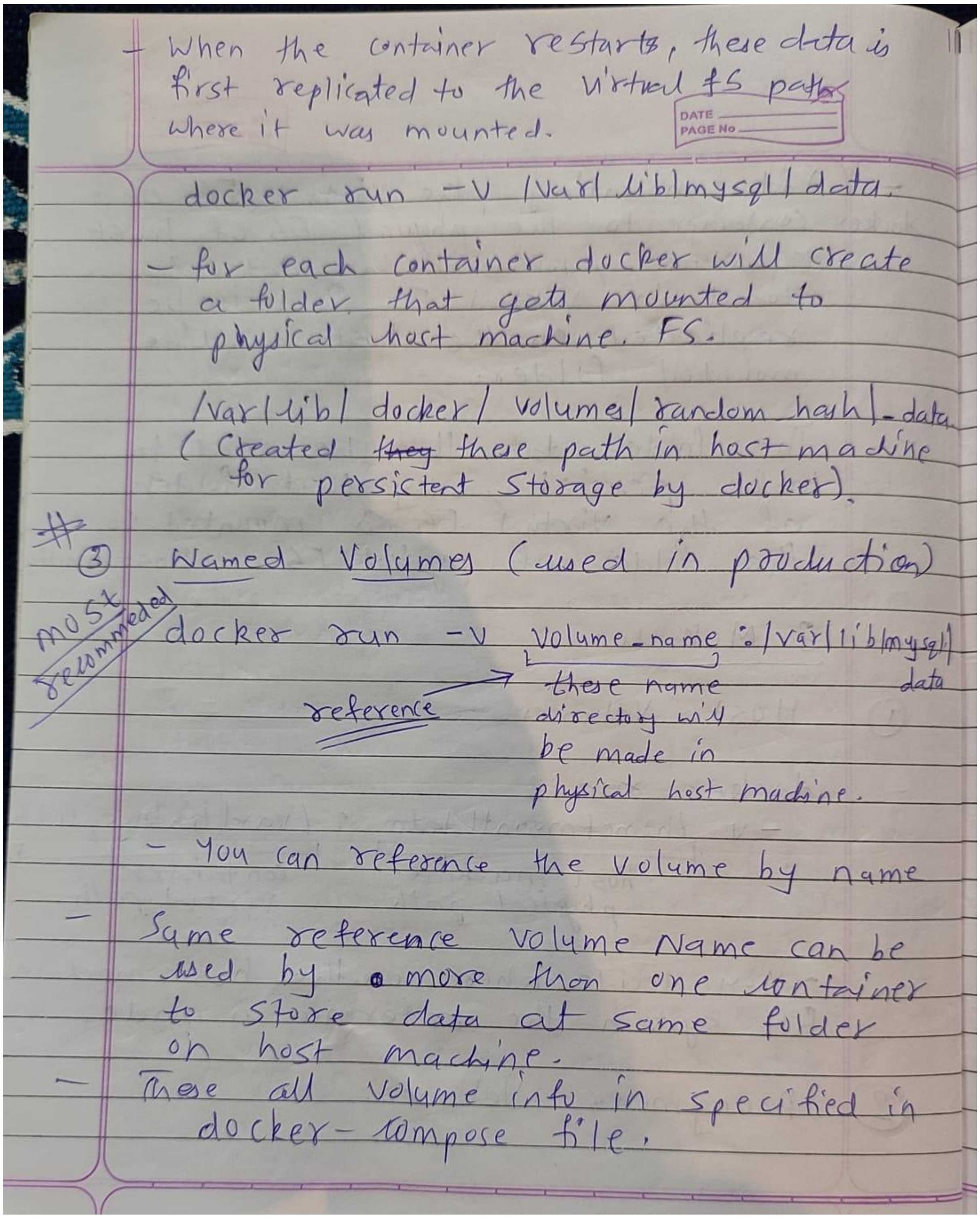
docker run

-v $\underbrace{\text{/home/mount/data}}_{\text{host machine physical path}} : \underbrace{\text{/var/lib/mysql/data}}_{\text{container virtual path.}}$

- You decide where on the host machine file system the reference is made.

② Anonymous Volumes.

DOCKER



- When the container restarts, these data is first replicated to the virtual FS path where it was mounted.

DATE
PAGE No

docker run -v /var/lib/mysql/data.

- for each container docker will create a folder that gets mounted to physical host machine. FS.

/var/lib/docker/volume/random hash/_data
(Created ~~they~~ these path in host machine for persistent Storage by docker).

③ Named Volumes (used in production)

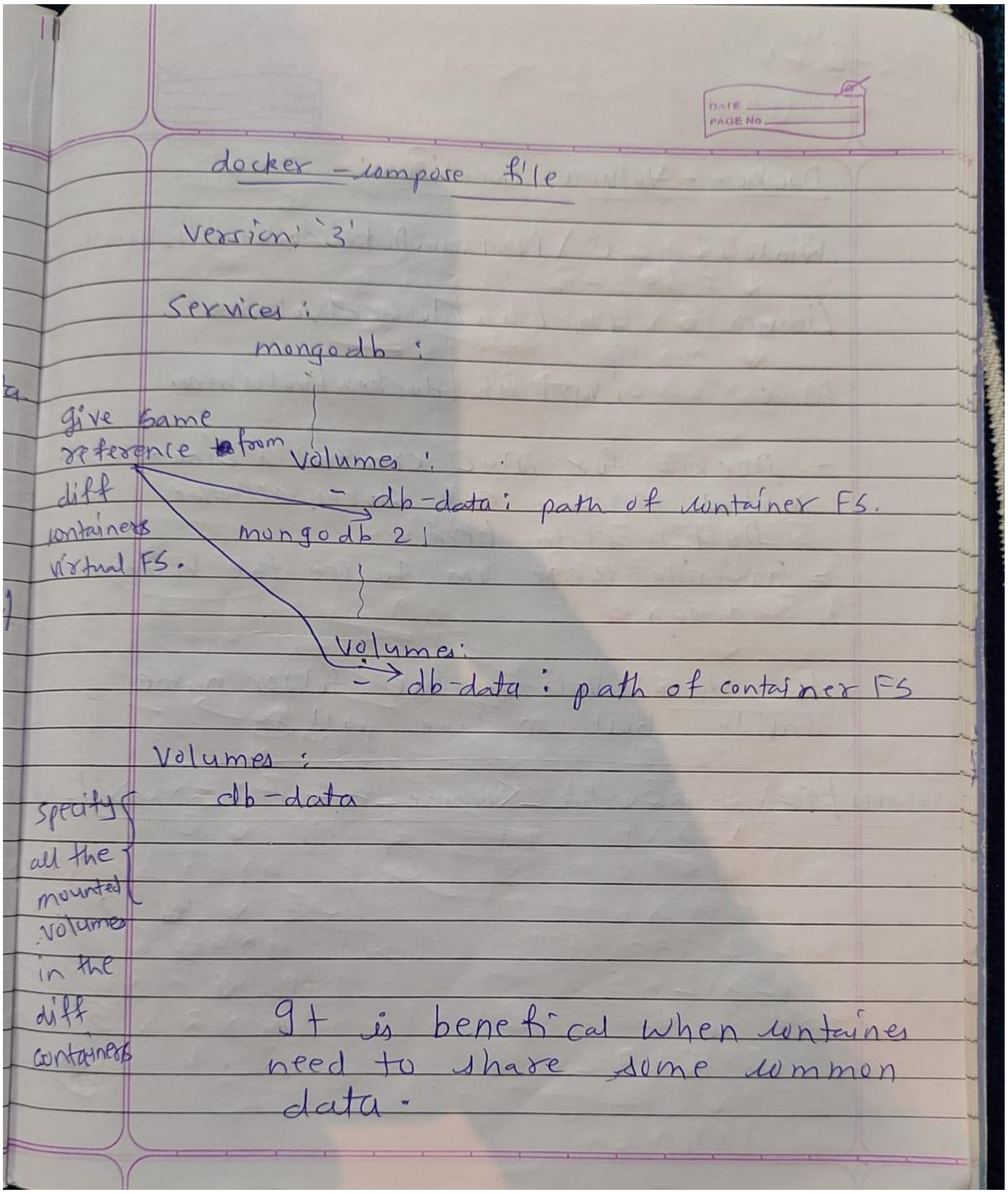
most recommended

docker run -v $\underbrace{\text{volume\_name}}$: /var/lib/mysql/data

reference $\longrightarrow$ these name directory will be made in physical host machine.

- You can reference the volume by name
- Same reference volume Name can be used by more than one container to store data at same folder on host machine.
- These all volume info in specified in docker-compose file.

DOCKER



docker-compose file

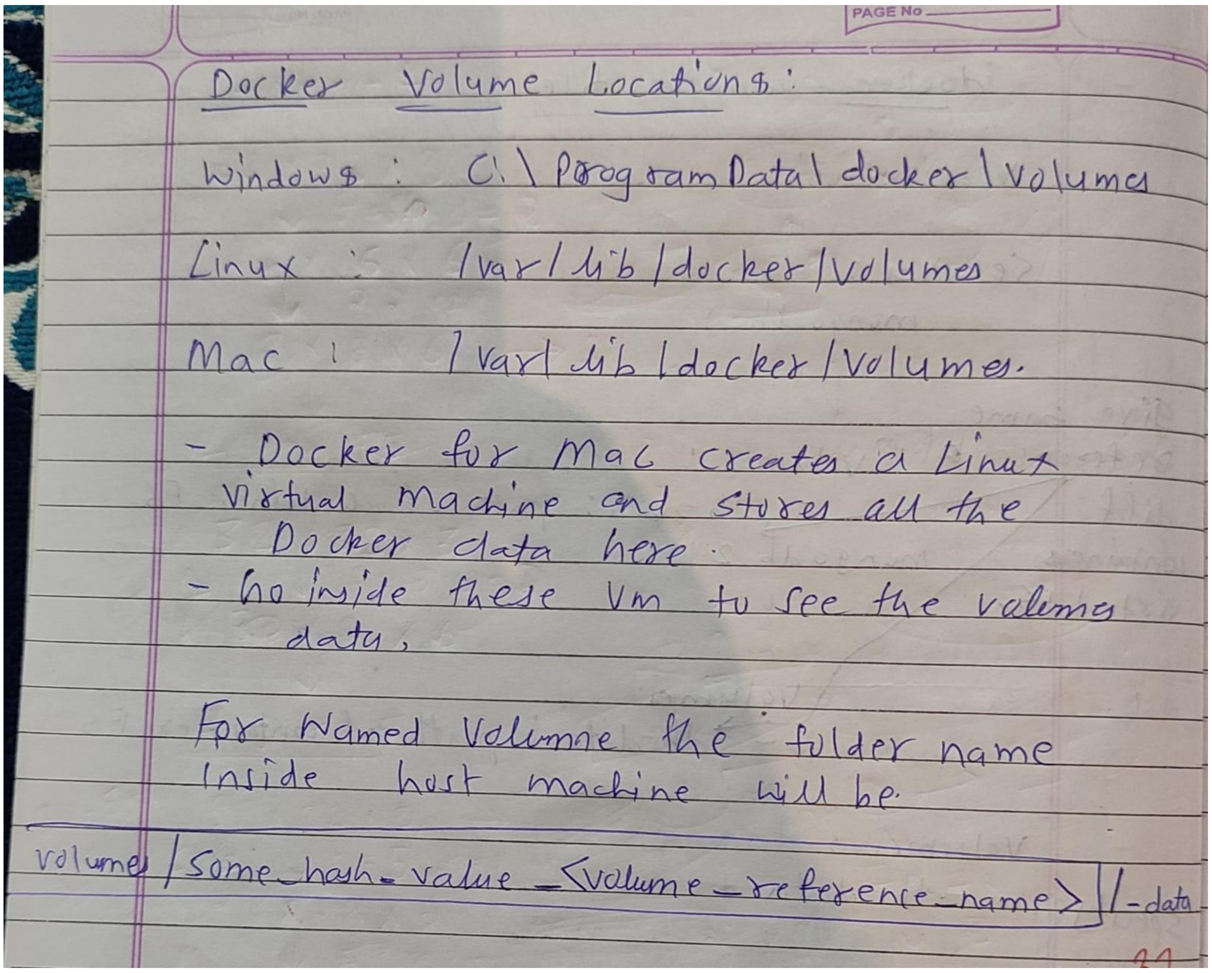
```
version: '3'

services:
    mongodb:
        |
        volumes:
          - db-data: path of container FS.
    mongodb 2:
        |
        volumes:
          - db-data: path of container FS

volumes:
    db-data
```

give same reference ~~to~~ from diff containers virtual FS.

specify all the mounted volumes in the diff containers

It is benefical when containes need to share some common data.

DOCKER



PAGE No

Docker Volume Locations:

Windows : C:\ProgramData\docker\volumes

Linux : /var/lib/docker/volumes

Mac : /var/lib/docker/volumes.

- Docker for Mac creates a Linux virtual machine and stores all the Docker data here.
- Go inside these vm to see the volumes data,

For Named Volumne the folder name inside host machine will be.

```
volumes/some_hash_value_<volume_reference_name>//_data
```