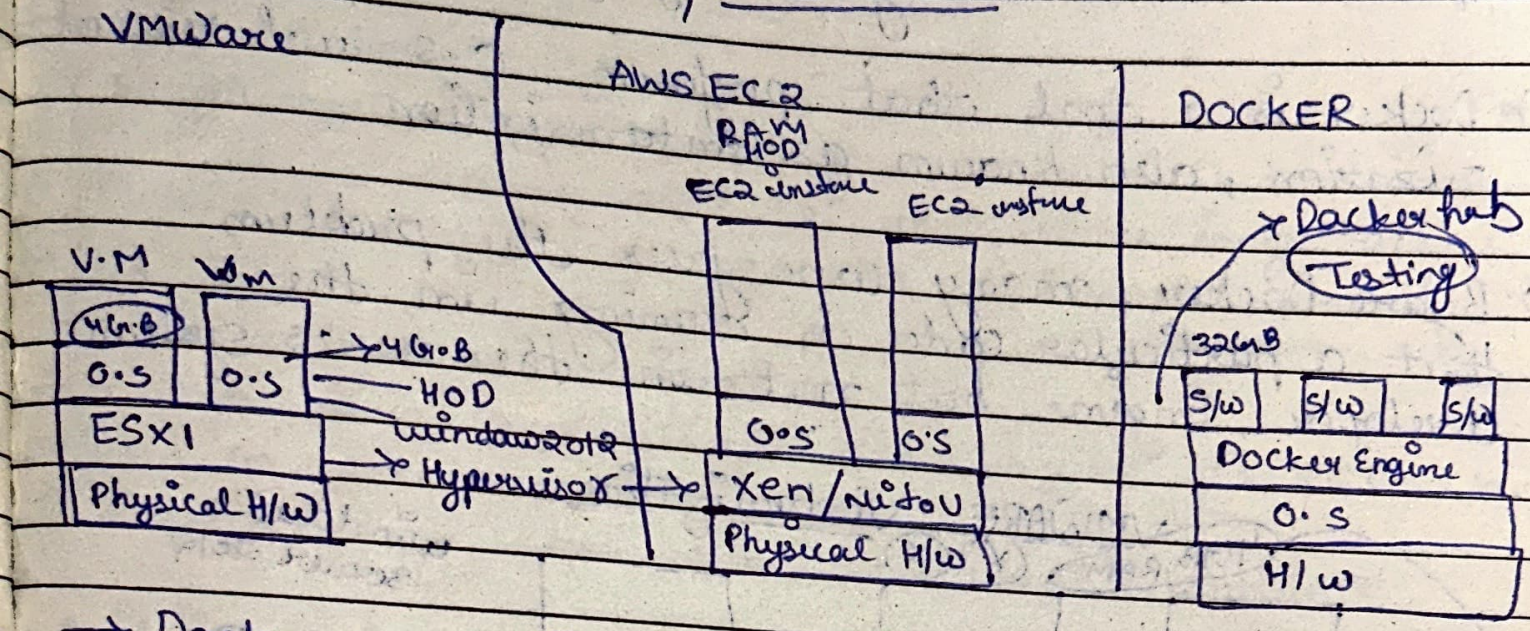
DOCKER :-



→ Docker was first release in march 2013. It is developed by solomen hykes and Sebastan Patil.

→ Docker is a set of ~~platf~~ platform as a Services that uses O.S level virtualization. Whereas Vmware uses hardware level virtualisation

→ Docker is an open source centeralized platform designed is Create, deploy and run applications

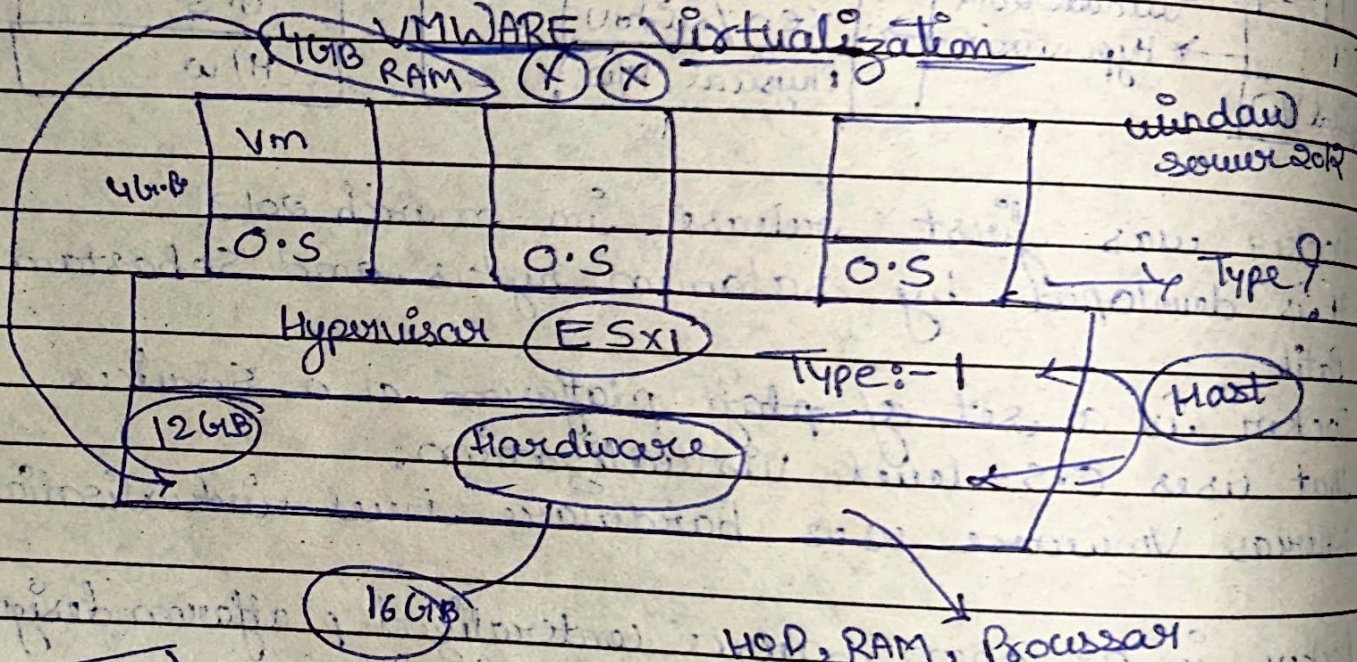
→ Docker uses Container on the host O.S. to run applic-ation it allows applications to use the same linux kernel as a System on the host Computer, Rather than creating a whole Virtual O.S.

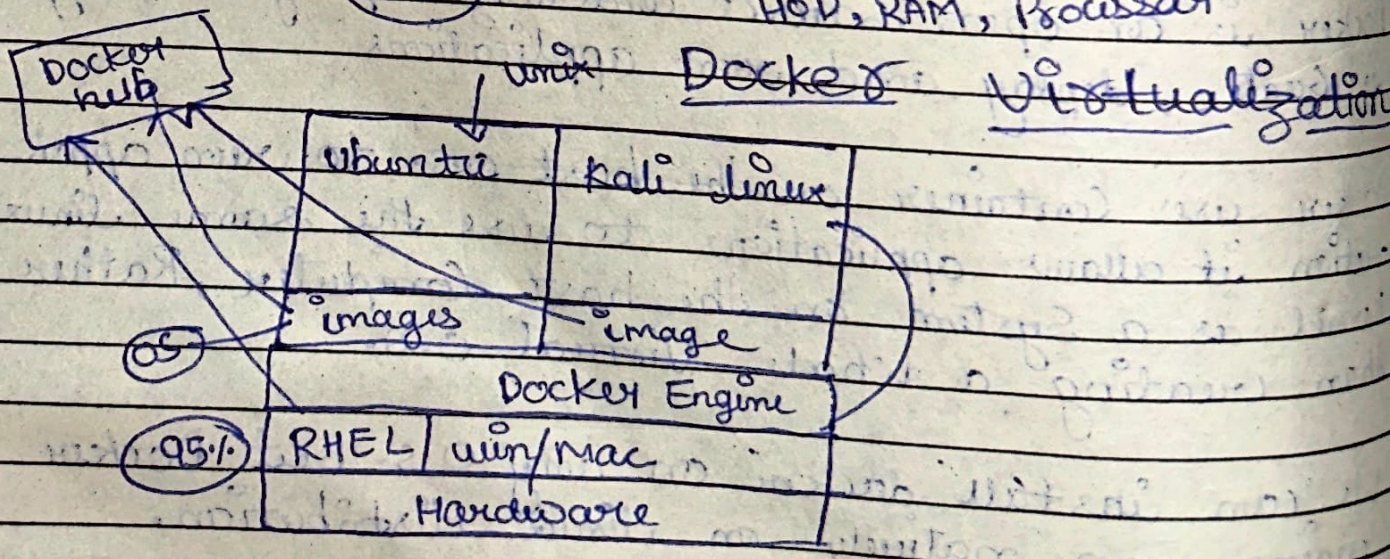
→ We can install docker on any O.S. but Docker engine run natively on linux distribution.

→ Docker written in 'go' language

→ Docker is a tool that performs O.S level virtualization, also known as containerization

→ Before Docker, many user faces the problem that a particular code is running in the developer Engine but not in the user's system





Page No.
Date

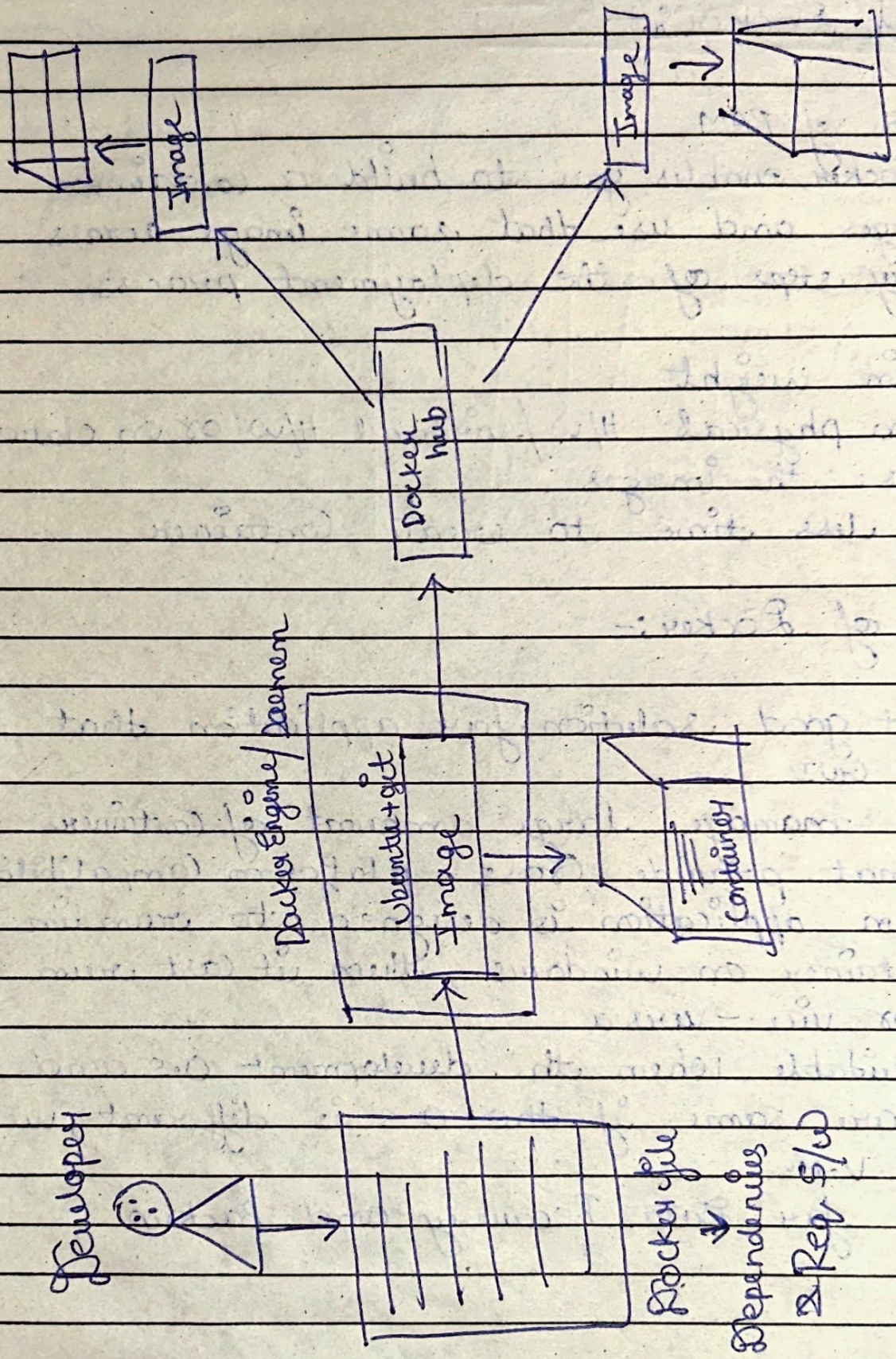
• Advantages of Docker :-

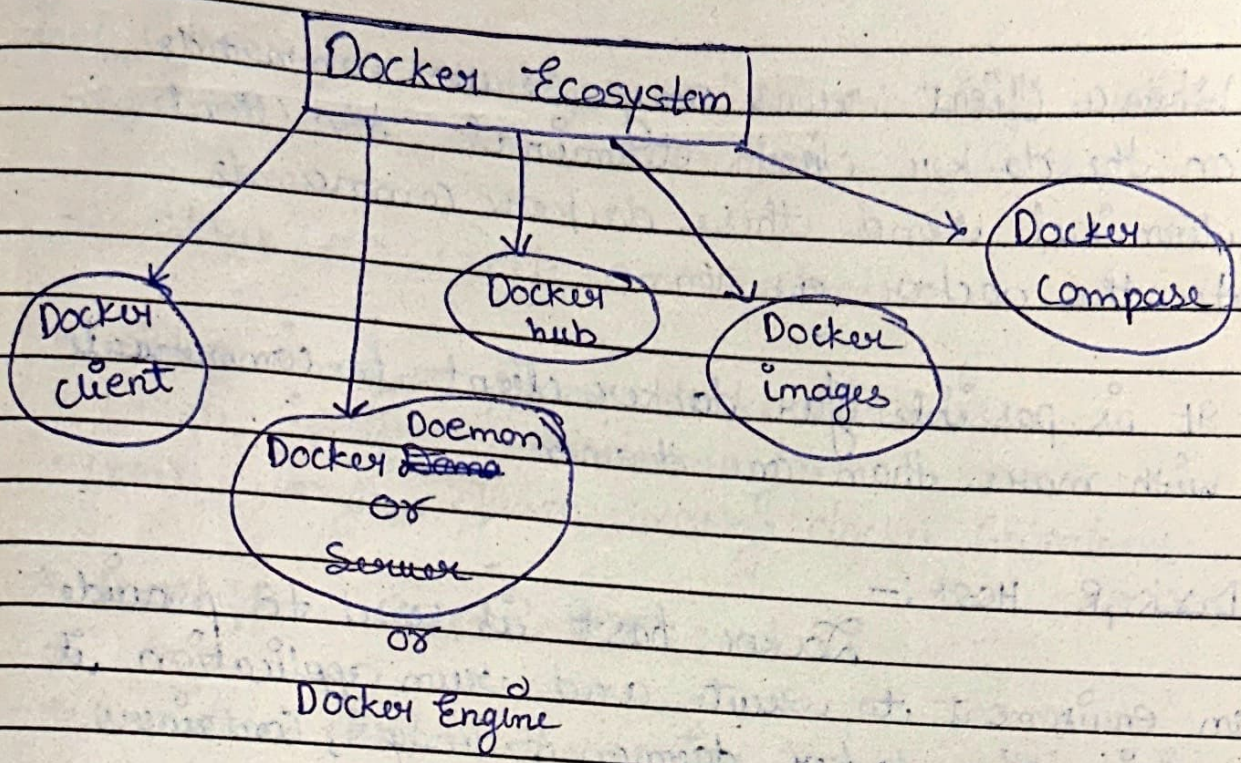
→ No pre-allocation of RAM
→ CI Efficiency :- Docker enables you to build a container images and use that same image across every steps of the deployment process
→ Less cost
→ It is light in weight
→ It can run on physical H/W / virtual H/W or on cloud
→ You can re-use the images
→ It took very less time to create container

• Disadvantages of Docker :-

→ Docker is not good solution for application that require rich GUI
→ Difficult to manage large amount of containers
→ Docker does not provide Cross-platform Compatibility means if an application is designed to run in a docker container on windows, then it cant run on linux or vice-versa.
→ Docker is suitable when the development O.S and testing O.S. are same if the O.S is different, we should use V.M.
→ No solution for Data Recovery and Backup

Architecture of Docker





Component of Docker :-

- **Docker Daemon :-**

→ Docker daemon run on the Host O.S.
→ It is responsible for runing containers to manges docker services
→ Docker daemon can communicate with other daemons

- **Docker Client :-**

→ Docker users can interact with docker through a client
→ Docker client uses commands and Rest API to communicate with the docker daemon

→ When a client runs any server commands on the docker client terminal, the client terminal send these docker commands to the docker daemon.

→ It is possible for docker client to communicate with more than one daemon.

- **DOCKER HOST :-**

Docker host is used to provide an environment to execute and run application it contains the docker daemon, Images, Containers network and storages.

- **Docker Hub / Registry :-**

Docker registry manages and stores the docker Engines.
There are two types of registries in the docker:

① Public Registry :- Public registry is also called as docker Hub

② Private Registry :- It is used to share images within the enterprise.

- **Docker images :-**

Docker images are the read only binary templates used to create docker Containers

or

Page No. _____
Date | | | |

Single file with all dependencies and configuration required to run a program

Ways to create an Images :-

- Take images from docker hub
- Create images from docker file
- Create images from existing docker containers

* DOCKER Containers :-

→ Container hold the entire packages that is needed to run the applications

or

→ In other words, we can say that, that images is a template and the containers is a copy of that template

→ Containers is like a virtual Machine

→ Images becomes Containers when the run on docker engine.

- Basics Commands in Docker :-

- To see all images present in your local machine

[ ]# docker images

- To find out images in docker hub

[ ]# docker search jenkins

- To download image from docker hub to local mac

[ ]# docker pull jenkins

- To give name to container

docker run -it --name shivam ubuntu /bin/bash

(run: create + start; -it: interactive mode; terminal)

- To check service is start or not

→ Service docker status

- To start Container

→ docker start shivam

- To go inside container

→ docker attach shivam

Page No.
Date

- To see all Container

→ docker ps -a

- To see only runing container

→ docker ps ← Process state

- To stop container

→ docker stop shivam

- To delete Container

→ docker rm shivam

DOCKERFILE COMPONENTS & Diff COMMANDS :-

- Login into AWS account and start your EC2 instance Access it from putty
- Now we have to create Container from our own image Therefore, create one Container first
- docker run -it --name shivam container ubuntu /bin/bash
- cd tmp1

• Now Create one file inside this tmp directory

→ touch myfile

• Now if you want to see the difference the base image & change on it then

→ docker diff shivamcontainer updateimage

O/p:- C /root
A /root/.bash_history
C /tmp
A /tmp/myfile

: D → Deletion
C → Change
A → Addition or Append

• Now create image of this container

→ docker commit newcontainer updateimage

→ docker images

• Now Create container from this image

→ docker run -it --name Katiyarcontainer updateima /bin/ba

root@id #ls
cd tmp/
tmp :# ls

O/p:- Myfile { you will get all file back }

DOCKERFILE :-

→ Dockerfile is basically a text file it contain some set of instruction

→ Automation of Docker image creation

DOCKER COMPONENTS

FROM → for base image. This command must be on top of the dockerfile

RUN :- To execute commands it will create a layer in image

MAINTAINER :- Author / OWNER / Description

COPY :- Copy files from local system (docker VM) we need to provide source / destination (we can't download file from internet and ~~only~~ any remote repo)

ADD :- Similar to COPY but, it provide a features to download files from internet, also we extract at docker images side

EXPOSE :- To expose ports such as port 8080 for tomcat, port 80 for nginx etc

WORKDIR :- To set working directory for a container

CMD :- Execute Commands but during Container creation

ENTRYPOINT :- Similar to CMD, but has higher priority over CMD, first Commands will be executed by ENTRY POINT only

ENV :- Enviroment Variable

- task 1 :-
  - Create a file named Dockerfile
  - Add instructions in Dockerfile
  - Build dockerfile and create image
  - Run image to create Container

vi Dockerfile

```
FROM Ubuntu
RUN echo "Technical gurugi" > /tmp/testfile
```

To create image out of dockerfile

```
docker build -t myimg .
```

```
docker ps -a
docker images
```

Page No.
Date

Now create container from the other images

docker run -it --name mycontainer myimg /bin/bash

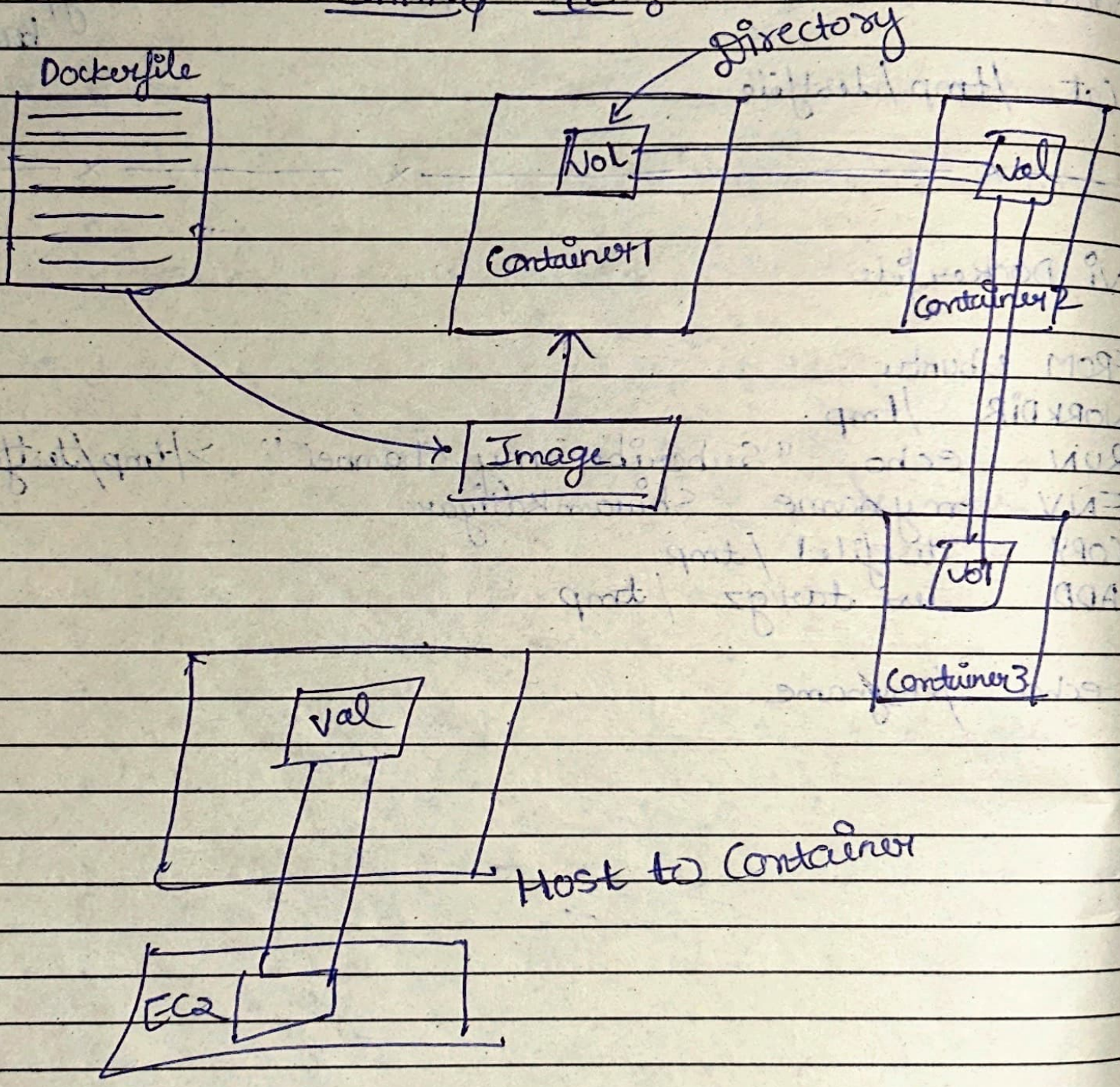
Cat /tmp/testfile

Vi Dockerfile

```
FROM Ubuntu
WORKDIR /tmp
RUN echo "Subcribe my channel" >/tmp/testfile
ENV myname ShivamKatiyar
COPY testfile1 /tmp
ADD test.tar.gz /tmp

echo $myname
```

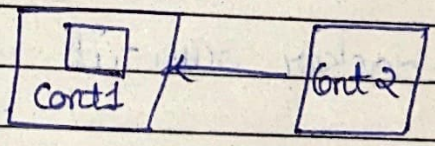
DOCKER VOLUMES & How to SHARE it :-

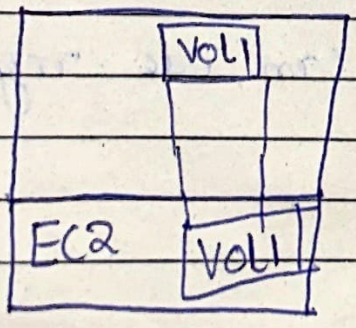


→ Volume is simply is directory inside our container
→ firstly, we have to declare this directory as a Volume and then share volume
→ Even if we stop container, still we can access volume
→ Volume will be created in one-Container
→ You can declare a directory as a Volume only while creating container
→ You can't create Volume from existing container
→ You can share one volume across any number of containers
→ Volume will not be include when you update an image

you can mapped volume in two ways:-

1. Container ←→ Container
2. Host ←→ Container





• Benifits of Volume :-

Decoupling container from storage

Share Volume among different container

Attach volume to containers

On deleting containers volumes does not delete

• Creating, Volume from Dockerfile :-

1. Create a Dockerfile and write

```
FROM ubuntu
VOLUME [ "/myvolume1" ]
```

• Then create image from this dockerfile

```
docker build -t myimage .
```

• Now Create a Container from the image &

```
docker run -it --name Container1 myimage /bin/bash
```

Now do ls, you can see myvolume1

- Now, share volume with another Container 1

docker run -it --name Container2 --privileged = true --volume-from Container1 ubuntu /bin/bash

Now after creating Container 2, myvolume 1 is visible whatever you do in one volume, can see from other volume

touch /myvolume1/Samplefile
docker start Container1
docker attach Container1
ls/my volume 1

you can see samplefile here

exit

Page No.
Date

- Now try to create Volume by using commands :-

→ docker run -it --name container3 -v /volume2 ubuntu /bin/bash

Do ls → cd /volume2

Now create one file cont3file and exit

Now create one more container, and share Volume 2

→ docker run -it --name Container 4 --privileged = true --volumes ~~other~~ from container 3 ubuntu /bin bash

OR

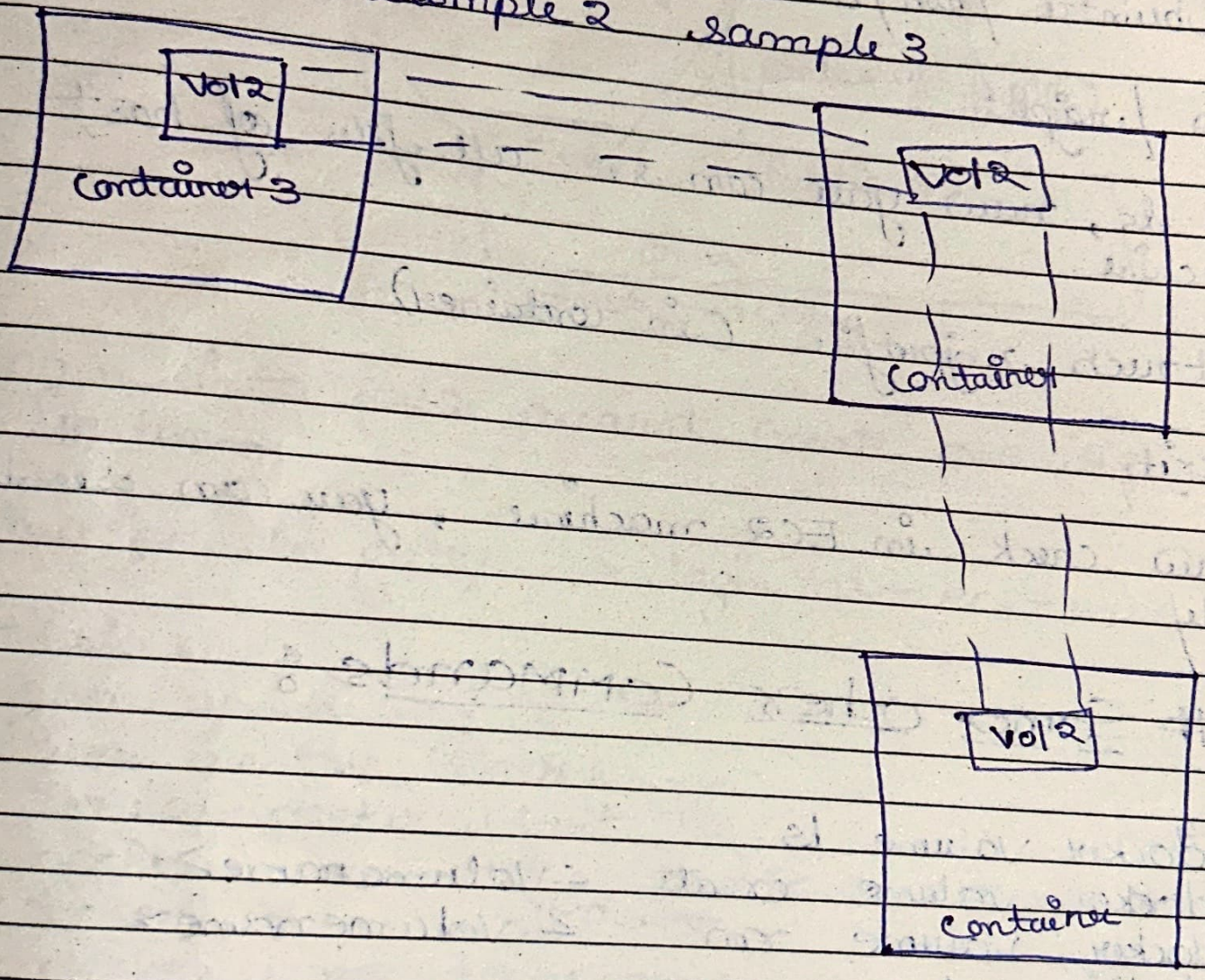
→ docker run -it --name Container 4 --privileged = true --volumes-from Container 3 ubuntu /bin/bash

Now you are inside container do ls, you can see volume 2

Page No.
Date

Now create one file inside the volume and then check in container 3, you can see that file

```
cd /volume2
ls
touch sample2 sample3
```



Page No.
Date

Volume (Host - Container)

→ Verify files in /home/ec2-user

→ docker run -it --name hostcont -v /home/ec2-user:/rajput --privileged=true ubuntu /bin/bash

→ cd /rajput

Do ls, now you can see all files of host machine

→ touch rajputfile (in container)

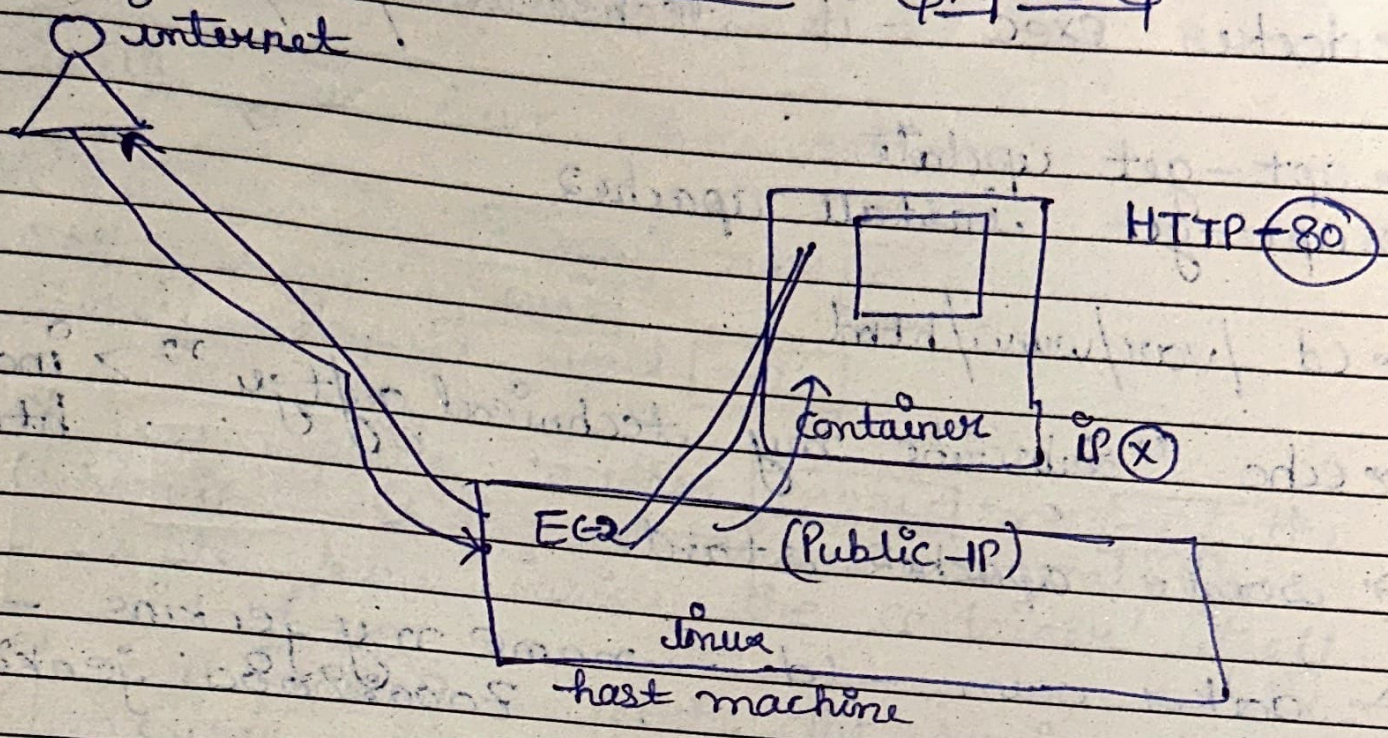
exit

Now check in EC2 machine, you can see the file.

Some other COMMANDS :-

→ docker volume ls
→ docker volume create <Volumename>
→ docker volume rm <Volumename>
→ docker volume prune
{it remaned all unused docker volume}
→ docker volume inspect <Volumename>
→ docker Container inspect <Volume name>

DOCKER PORT EXPOSE



Login into AWS account create one linux instance

Now go to putty → login as - ec2-user

→ sudo su
→ yum update -y
→ yum install docker -y
→ service docker start
→ docker run -td techserver -p 80:80
→ docker ps
→ docker port techserver
o/p :- 80/TCP → 0.0.0.0/80

-td → daemon

host container port or publish

→ docker exec -it techserver /bin/bash

→ apt-get update
→ apt-get install apache2

→ cd /var/www/html

→ echo "Subscribe my technical gyttar" > index.html

→ service apache2 start

→ docker run -td --name my jenkins -p 8080:8080 jenkins

• Difference Between docker attach and docker echo

Docker exec creates a new process in the containers enviroment while docker attach just connect the standard Input/Output of the main process inside the container to corrusponding standard input/output error of current terminal

Docker exec is specifically for runing new things in a already started containers, be it a shell or some other process

Page No.
Date

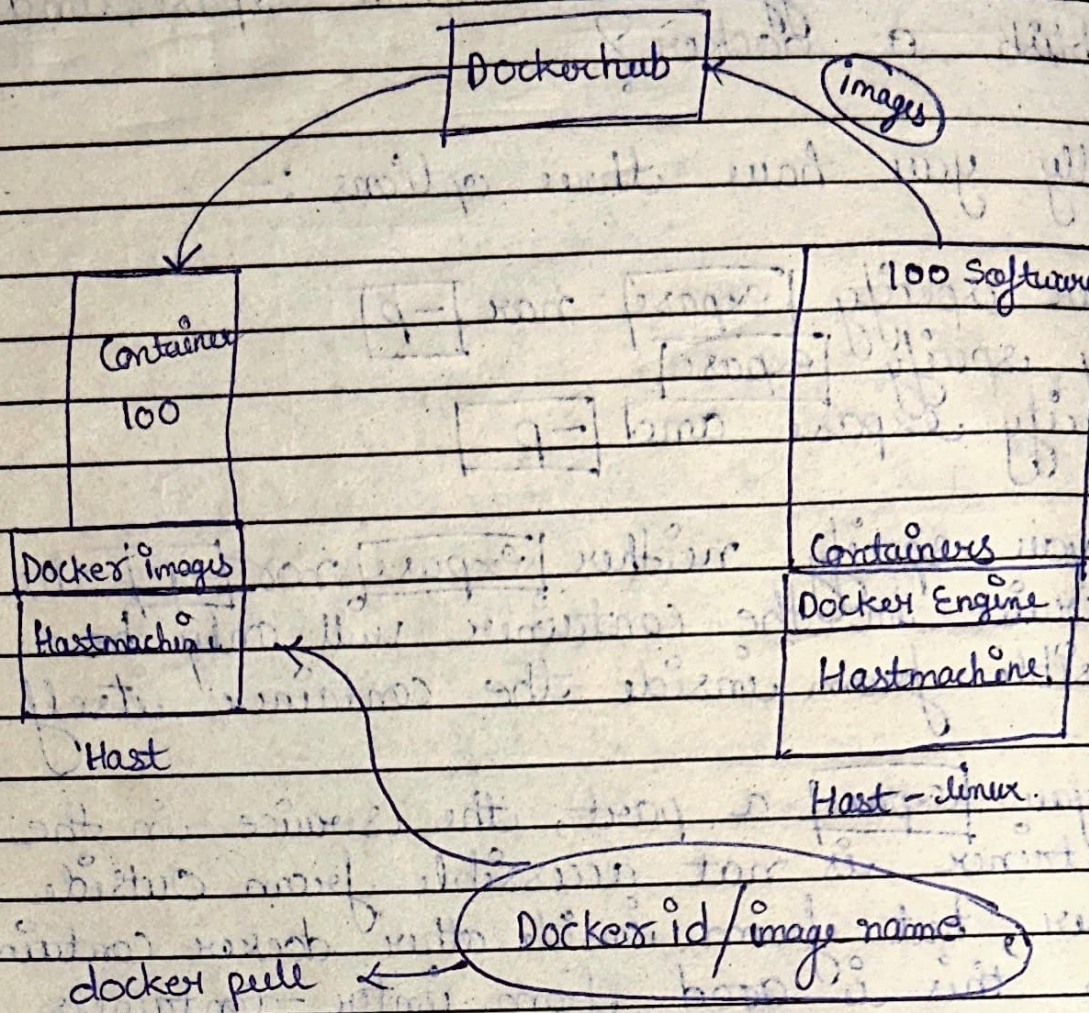
Ques:- What is difference between expose and publish a docker?

Basically you have three options :-

1. Neither specify `expose` nor `-p`
2. Only specify `expose`
3. Specify `expose` and `-p`

- if you specify neither `expose` nor `-p`, the service in the container will only be accesible from inside the container itself
- if you `expose` a port, the service in the container is not accessible from outside docker, but from inside other docker container so this is good from inter-containers communication
- if you `expose` and `-p` a port, the service in the container is accessible from anywhere even outside docker
- if you do `-p` but do not `expose` docker does an implict expose
  This is because, if a port is open to the public, it is ~~a port~~ automattically also open to the other docker containers.
  Hence '-p' include expose

• How to push docker images on docker hub :-



→ Go to AWS account → select Amazon linux Nau
go to putty → login as - ec2 - user

→ Sudo su
→ yum update -y
→ yum install docker -y
→ Service docker start
→ docker run -it ubuntu /bin/bash
Now create some files inside container
Now create images of this container

→ docker commit Container1 image1

Page No. _____
Date

Now create account in hub.docker.com

→ docker login
enter your username and password

Now give tag to your image
docker tag image1 dockerid/newimage
docker push dockerid/newimage

Now you can see the image in dockerhub
Now create one instance in tokyo region and pull image from hub

→ docker pull dockerid/newimages

→ docker run -it --name mycon dockerid/image /bin/bash

Some important commands :-

① Stop all runing Container : docker stop $(docker ps -a -q)

② Delete all stoped container:-
docker rm $(docker ps -a -q)

③ Delete all image :-
docker rmi -f $(docker images -q)