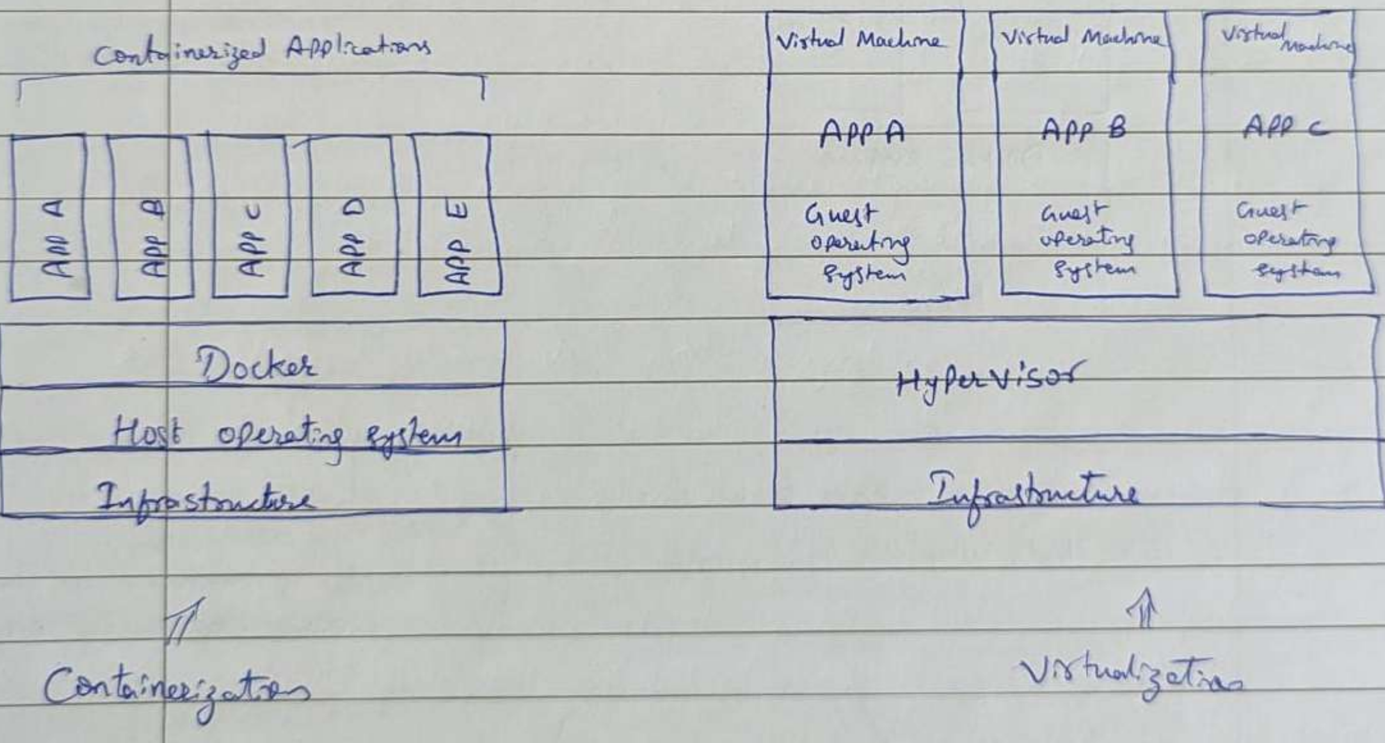
Q What is a container?

=> A container is a standard unit of software that packages up code and all it's dependencies so the application runs quickly and reliably from one computing environment to another.

A docker container image is a lightweight, standalone, executable ~~package~~ package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

A container is a bundle of Application, Application libraries required to run your application and the minimum system dependencies.



Q) Why are containers light weight?

=> Containers are light weight because they use a technology called containerization, which allows them to share the host operating system's kernel and libraries, while still providing isolation for the application and it's dependencies. This results in a smaller footprint compared to traditional virtual machines, as the containers do not need to include a full operating system. Additionally, Docker containers are designed to be minimal, only including what is necessary for the application to run, further reducing their size.

Let's try to understand this with an example:

Official ubuntu base image which you can use for you container, it's just ~22MB , ~~On~~ isn't it very small ? on a contrary if you look at official ubuntu VM image it will be close to ~2.3GB. So the container base image is almost 100 times less than VM image.

Virtualization :-

Virtualization is the creation of a virtual --rather than actual-- version of something, such as an operating system, a server, a storage device or network resource.

→ A virtual machine is an isolated environment which has an operating system of its own. It can contain either one or more applications inside it.

→ A single physical host can run either one or more virtual machines.

→ Virtual machines virtualize both OS as well as hardware of the real machine

→ Each VM is completely independent of each other.

→ These multiple virtual machines can run several operating systems and applications on just a single physical server.

Docker

→ Docker is an open-source centralised platform designed to create, deploy and run applications.

→ Docker uses containers on the host O.S to run applications. 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 runs natively on linux distribution.

→ Docker written in 'go' language.

→ Docker is a tool that performs OS level virtualization, also known as ~~centrali~~ containerization

→ Before Docker, many users faces the problem that a Particular code is running in the developer's system but not in the User's System.

Q) Does container actually have OS ?

=> 5% files of OS will be there in container in the form of image. but compared to Actual OS it's negligible, that's why we say container doesnot have O.s.

~~But if we say~~

But actually container have OS, it's 5% only but it has.

__/__/____

Q Why Docker ?

⇒ → Docker does not possess any dependency problems. If it builds in your system, it can build anywhere.

→ Multiple environment for production, QA, testing etc., can be created in minutes.

→ Implement reliable CI easily.

→ The container image is an executable which contains all the files and dependencies required to run the application. Hence, creating an environment can be as easy as running an executable file.

→ Images are bigger, but they are broken down into layers.

→ Changes are updated by just updating the layer which underwent change.

→ It reduces the cost overhead by saving disk, network and memory usage.

Advantages of Docker

→ No pre-allocation of RAM.

→ CI efficiency : Docker enables you to build a container image and use that same image across every step of the deployment process.

→ less cost.

→ It is light in weight.

→ It can run on Physical H/W (or) virtual H/W (or) on cloud.

→ you can re-use the image.

→ It take very less time to create container.

__/__/____

=> we move image only not container to other computing environment

→ light weight means → it requires less resources ~~for run~~ to run the application

=> We can make changes in containers but we cannot make changes in image.

* Disadvantages of Docker:

→ Docker is not a ~~application~~ good solution for application that requires rich GUI.

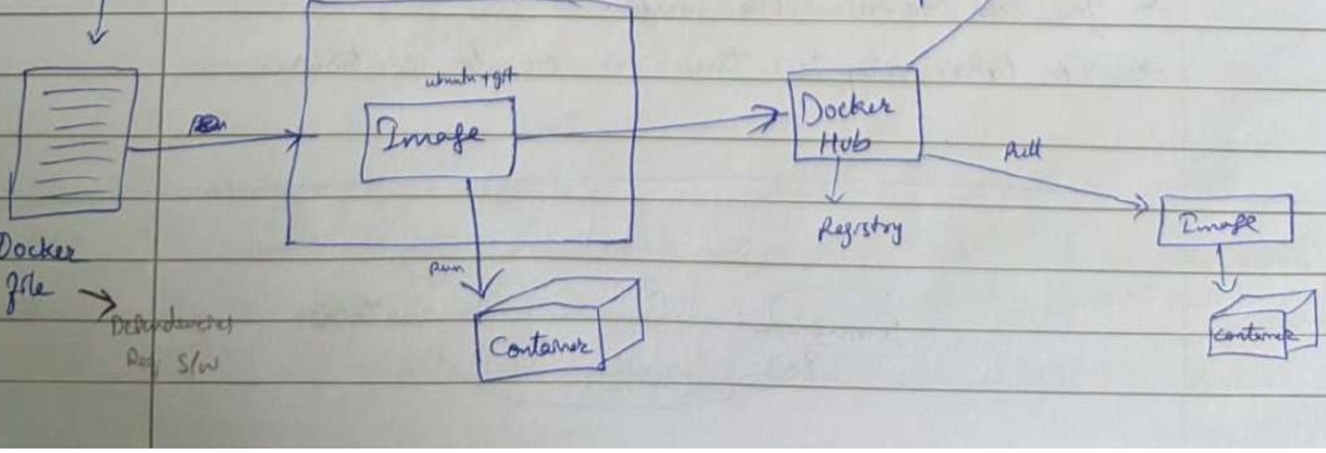
→ Difficult to manage large amount of containers

→ Docker does not provide cross-platform compatibility means if an application is designed to run in a docker container on ~~which~~ windows, then it can't run on linux or vice-versa.

→ Docker is suitable when the development OS and testing OS are same, if the OS is different, we should use VM.

→ No solution for Data Recovery and Backup.

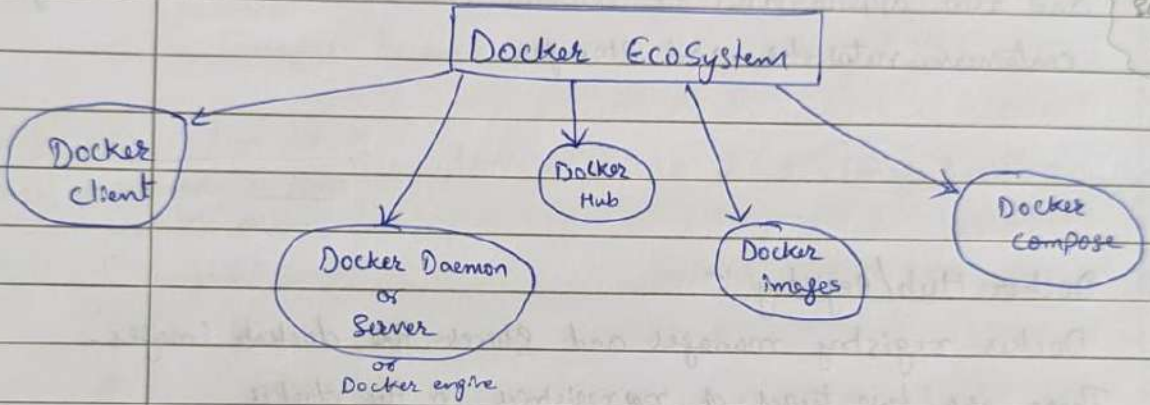
* Docker Architecture:



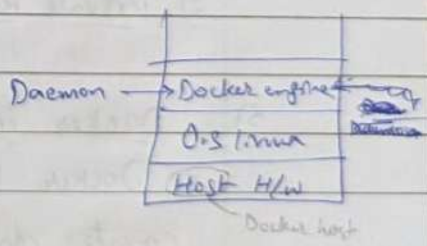
__/__/__

→ containers are layed file system
layer1 what software is mentioned will get installed first then layer2 what is mentioned here will get installed. then layer 3

Ecosystem means set of software ~~and~~ or packages



* Components of Docker



1) Docker Daemon :-

→ Docker ~~D~~ daemon runs on the Host O.S

→ It is responsible for running containers to manages docker services

~~⊕ Docker Dea~~

→ Docker Daemon can communicate with other daemons

2) Docker Client :

Docker ~~user~~ users can interact with docker daemon through a client (CLI)

→ Docker client uses commands and Rest API to communicate with the docker daemon.

→ When a client run any server command on the docker client terminal, the client terminal sends these docker commands to the docker daemon.

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

3) Docker Host :-

Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks and storages.

Docker host is physical ~~hard~~ hardware where you are creating all containers

4) Docker Hub/Registry :-

Docker registry manages and stores the docker images.

These are two types of ~~reg~~ 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.

5) Docker images :-

→ Docker images are the read only binary templates used to create docker containers.

(or)

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

* Ways to create an Images

① Take image from docker hub.

② Create image from docker file.

③ Create image from existing docker containers

6) Docker container :-

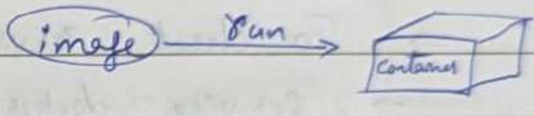
→ Container hold the entire package that is needed to run the application.

(Or)

In other words, we can say that, the image is a template and the container is a copy of that template

→ Container. is like a Virtual machine.

→ Images becomes container when they run on docker engine.



Basic commands .

* Install Docker

```
Sudo apt update
Sudo apt install docker.io -y
```

* Basic commands in Docker

⇒ To see all images present in your local machine.

```
→ [    ]# docker images
```

⇒ To find out images in docker hub.

```
→ [    ]# docker search jenkins
(or)
→ [    ]# docker search ubuntu
```

mention what you want from docker hub.

___/___/_____

⇒ To download image from dockerhub to local machine

→ [ ] # docker pull jenkins

⇒ To give name to container

~~[~~] → docker~~

→ docker run -it --name Ahtesham ubuntu /bin/bash

name of container

image name

~~docker hub~~

terminal

interactive mode

Create & start

⇒ To check, service is start or not

→ service docker status → press Ctrl + Z to come out from that prompt

(or)

→ Systemctl status docker

To start docker service → Systemctl start docker.

→ service docker start.

⇒ To start contaner

→ docker start Ahtesham → container name

⇒ To go inside contaner → for this contaner should be started first.

→ docker attach Ahtesham → container name.

⇒ To see all contaners

→ docker ps -a

⇒ To see only running contaners

→ docker ps → process status

→ To stop contaner

→ docker stop Ahtesham → contaner name.

You should get exit from container first, then stop the container

__/__/__

⇒ To delete contaner

→ docker rm Ahtesham — contaner name

(rm → remove)

⇒ To check where docker is there

→ which docker.

or

you can check docker version also

→ docker --version.

youtube ⇒ ~~docker info~~

⇒ To check all details of docker in server & memory — which OS it's using all details get here

→ docker info

⇒ The daemon service for docker is 'docker' which should be enabled ~~first~~ and started in order to start working with docker you can check status by

systemctl status docker

⇒ Enable and start Docker

→ systemctl enable docker

→ systemctl start docker

login into contaner

⇒ docker run -it centos /bin/bash

[root@ contanier id] # cat /etc/os-release

you get details about image. Which os it has ~~etc~~ and all.

Enable & start docker

systemctl enable docker

systemctl start docker

systemctl status docker

☆

__/__/__

=> Login into AWS account and start your EC2 instance. Access it from putty.

→ Now we have to create contaner from our own image.

Therefore, create one contaner first

→ docker run -it --name bhupicontainer ubuntu /bin/bash

* cd temp/

Now create one file inside this temp directory

→ touch myfile   → creating this file, when I ~~[illegible]~~ create new image from this contamer, in new contamer to verify the file will be there or not.

Now if you want to see the difference between the base ~~image~~ image ~~and~~ and changes on it then

→ docker diff bhupicontainer ~~updateimage~~
(container name)

```
o/p =>  C  /root
        A  /root/.bash_history
        C  /temp
        A  /temp/myfile
        D
```

D → deletion
C → changes
A → Append (or) Addition

Now, create image of this Container

→ docker commit ~~newcontainer~~ bhupicontainer updateimage
(container name) → give image name here which you are creating new
↳ Creating image from container

→ docker images

Now create contaner from this image

→ docker run -it --name rajcontainer updateimage /bin/bash

```
root@cid # ls
   "    # cd temp/
     temp # ls
o/p => myfile
```

{ you will get all files back }

* Docker file

→ Docker file is basically a text file. It contains some set of instructions

→ Automation of Docker image creation.

Docker components.

FROM → For base image. This command must be on top of ~~docker file~~ 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 any remote repo)

ADD → Similar to copy but, it provides a feature to download files from internet, also we extract files at docker image side.

EXPOSE → To expose ports such as port 8080 for tomcat, port 80 for nginx etc.

WORKDIR → To set working directory for a container creation.

→ It's basically identification that source code is going to save in this working directory. → example WORKDIR /app
/app

Dockerfile → D → should be capital only.

__/__/____

~~ENTRYPOINT~~ →

CMD → Execute commands but during container creation.

→ here values can get ovrdeable.

ENTRYPOINT → Similar to CMD, but has higher priority over CMD, first commands will be executed by ENTRYPOINT only.

→ will have non overridable value

ENV. → Environment Variables.

ARG →

Dockerfile

capital D

1) → Create a file named Dockerfile
2) → Add instructions in Dockerfile
3) → Build dockerfile to create image
4) → Run image to create container

```
vi Dockerfile
FROM ubuntu: xx
RUN echo "Md Ahtesham" > /tmp/testfile
```

→ mention version if needed

TO create image out of docker file.

docker ~~build~~ build -t myimg .

tag

→ docker ps -a

→ docker images.

Now, create container from the above image.

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

→ cat /tmp/testfile

Example to create Dockerfile.

```
vi Dockerfile

FROM ubuntu
WORKDIR /tmp
RUN echo "This is Md Ahtesham" > /tmp/testfile
ENV myname Nadeem
COPY testfile1 /tmp
ADD test.tar.gz /tmp

:wq!
```

$myname → by this you can find in docker

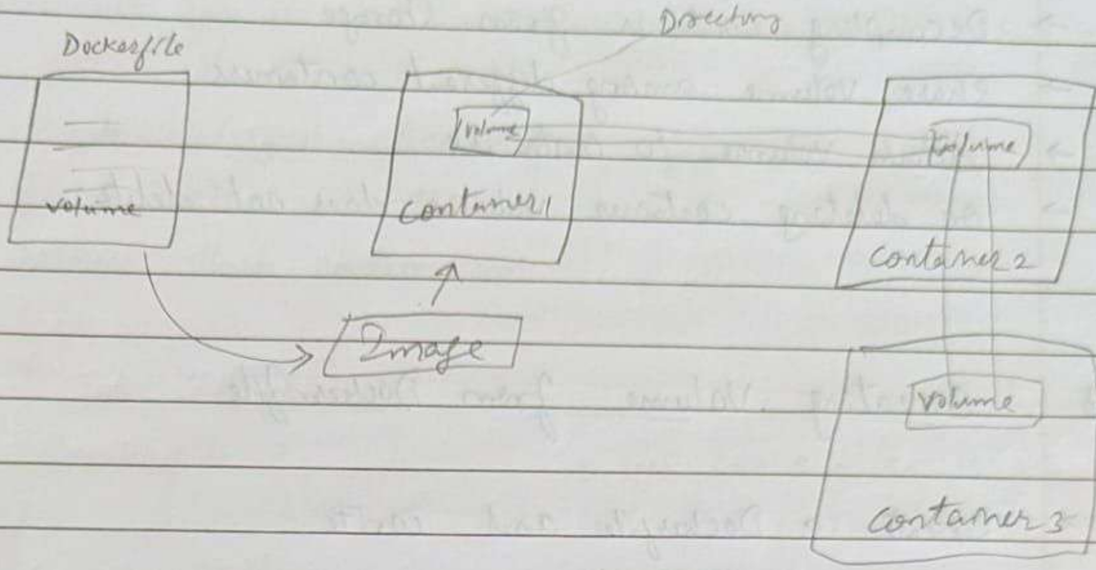
```
# ls
Dockerfile
# touch testfile1
# ls
Dockerfile testfile1
# touch test
# ls
Dockerfile ~~testfile~~ ~~test~~ test testfile1
```

```
# tar -cvf test.tar test
# ls
Dockerfile ~~testfolder~~ test testfile1 test.tar

# gzip test.tar
# ls
Dockerfile test testfile1 test.tar.gz
# rm -rf test
# ls
Dockerfile testfile1 test.tar.gz

# docker build -t newimage .

# docker ~~run~~ images.

# docker run -it --name newcontainer newimage /bin/bash

root@cid:/tmp # ls
    test testfile testfile1

root@cid:/tmp # cat testfile
This is Md Ahtesham

root@cid:/tmp # echo $myname
Nadeem
```

WORKDIR → means when container run, what you give in WORKDIR → container opens in that folder.

Docker Volumes and How to Share it

__/__/__



→ If we delete container, volume won't get deleted with container, we have to delete volume separately.

→ We can share volume from container to container and Host to container also.

→ Volume is simply a directory inside our container.

→ Firstly, we have to declare this directory as a volume and then share volume.

→ Even ~~we~~ if we stop container, still we can access volume.

→ Volume will be created in one container and it will share with other contain

→ 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 ~~from~~ across any number of containers

→ Volume will not be included when you update an image.

→ You can Mapped volume in two ways.

① container ⟷ containers

② Host ⟷ containers

if they add in volume of other Containers we can also see th

If volume is shared between contain

Benefits of volume

→ Decoupling container from storage

→ Share volume among different containers

→ Attach volume to containers

→ On deleting container volume does not delete.

Creating Volume from Dockerfile

⇒ Create a Dockerfile and write

```
vi Dockerfile
FROM ubuntu
VOLUME ["/myvolume"]
                          :wq!
```

Then create image from this dockerfile

```
→ docker build -t myimage .
```

Now create a container from this image and Run

```
→ docker run -it --name container1 myimage /bin/bash
```

Now do ls you can see myvolume

→ exit

Now, Share volume with another container

Container ⟷ container

```
→ docker run -it --name container2 --privileged=true
  --volumes-from container1 ubuntu /bin/bash
```

↳ container2 image.

Now after creating container 2, myvolume is visible.
whatever you do in one Volume, can see from another volume.

→ touch /myvolume/samplefile exit from container 2
→ docker start container 1
→ docker attach container 1
→ # ls
# cd /myvolume
Samplefile → you can see sample file here.

exit.

* Now, try to create volume by using command

→ docker run -it --name container 3 -v /volume2 ubuntu /bin/bash
subdirectory

Do ls → cd /volume2
Now # touch cont3file
exit

Now create one more container and share volume2

→ docker run -it --name container4 --privileged = true ~~-evolumes-from~~ container3 ubuntu /bin/bash
-- volume-from ✓

__/__/___

Now you are inside contaner4, do ls , you can see Volume 2 .

Now create one file inside this volume and then check in contaner 3 , you can see that file.

Volumes (Host ⟷ Contaner)

[illegible]

→ verify files in /home/ec2-user

→ touch file1 file2 file3

[illegible] ec2-user

volume

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

host

Contaner name

→ cd /rajput

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

→ touch rajputfile (in contaner)

exit

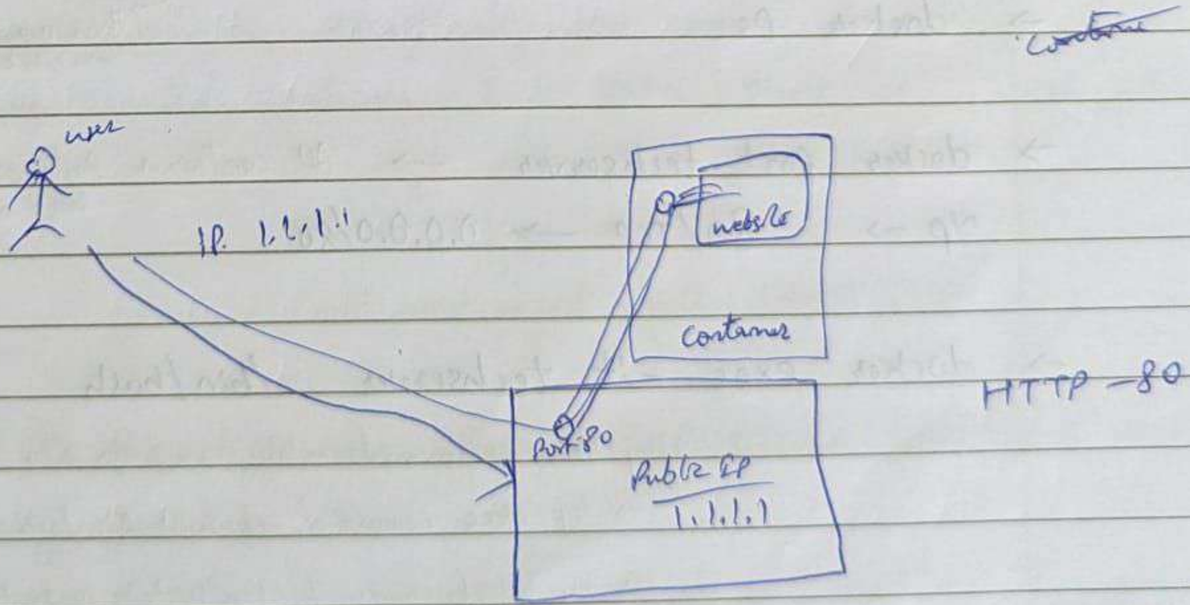
Now check in Ec2 machine, you can see the files

__/__/____

Some other commands for Volumes

```
→ docker volume ls
→ docker volume create <volume name>
→ docker volume rm <volume name>

→ docker volume prune
      {It removed all unused docker volume}

→ docker volume inspect <volumename>
→ docker contaner inspect <contaner name>
```

* Docker port Expose



★ Project

/bin/bash → we can go into the container
that's y it's used in run command

Login into Aws account create one ~~ubuntu~~ Amazon Linux instance

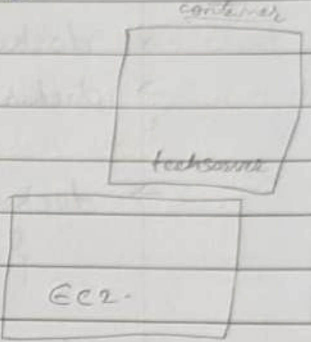
Now go to putty → login as → ec2-user

→ Sudo su

→ ~~apt~~ yum update -y

→ yum ~~apt~~ install docker.io -y

→ Service docker start



in Security Allow 80 port HTTP in firewall

→ ~~docker run -td --name 1 techserver~~

O/P ⇒ ~~80/TCP~~ → 0.0.0.0/80

80:80 → host port and container port are mapped

→ ~~docker exec -it techserver /bin/bash~~

→ docker run -td --name techserver -P 80:80 ubuntu

-td → daemon

-P → Port or publish

80:80 → host port, container port

Example to run inside container after building docker file. just for understanding

→ docker run -it --name md -p 8000:8000 myimage → this will directly take into the container

-P ⇒ it will directly expose also

→ docker ps

→ docker port techserver → All container Ports which are mapped to host are showed by that command

O/P ⇒ 80/TCP → 0.0.0.0/80

→ docker exec -it techserver /bin/bash

→ by this command also we go into container

→ by exec command it will start a new Process in container

→ by attach command it will take to existing process in container

⇒ docker attach containername

Inside container do below commands

→ apt-get update

→ apt-get install apache2 -y

in ubuntu we use apache server, in ubuntu we give it as apache2

now I can create web server

__/__/____

→ cd /var/www/html

→ echo "My name is Md Ahtesham" > index.html

→ Service apache2 ~~restart restart~~ restart

~~→ Service apache2 Start~~ open chrome => ip address paste and enter get output here

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

↗ => In ec2 firewall Allow custom TCP 8080 Port

To access jenkins wrote this

open chrome → ipaddress : 8080 => you see jenkins page

exec → execute

Q Difference between docker attach and docker exec?

=> Docker exec creates a new process in the container's environment while docker attach just connect the standard Input/output of the main process inside the container to corresponding standard input/output error of current terminal.

docker exec is specifically for running new things in a already started container, but it a shell or some other process

pid → process id

ppid → parent process id

Q What is the difference between expose and publish a docker?
-p

=> Basically you have three options :-

1) Neither specify expose nor -p

2) only specify expose

3) Specify expose and ~~-p -p~~ -p

① If you specify neither expose nor -p, the service in the container will only be accessible from inside the container itself.

(2) 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 for inner-contaner communication.
If you do -p but do not expose docker does an implicit expose. This is because, if a port is open to the public, it is automatically also open to the other docker containers Hence '-p' includes expose.

(3) If you expose and -p a port, the service in the container is accessible from anywhere, even outside docker.

project

# How to push docker image in dockerhub

→ Go to Aws account → select Amazon linux

Now go to putty → login as — ec2-user

→ sudo su
→ yum update -y
→ yum install docker -y

→ systemctl start docker

→ docker run -it ubuntu /bin/bash

Now create some file inside contaner
Now create image of this contaner.

__/__/____

→ docker commit container1 image1

Now Create account in hub.docker.com

Now go to EC2 instance

→ docker login
  enter your username and password

Now give tag to your image
dockeraccount id
anyname

→ docker tag image1 dockerid/newimage
imagename

→ docker Push dockerid/newimage

Now you can see this image in docker hub ~~acco~~ account.
Now create one instance in tokyo region. and pull image from hub.

→ docker pull docker/newimage

→ docker run -it --name mycontainer dockerid/newimage /bin/bash

→ docker images.
  dockerid/newimage.

__/__/__

* Some important commands

① Stop all running containers :-

→ `docker stop $(docker ps -a -q)`

② Delete all stopped containers :-

→ `docker rm $(docker ps -a -q)`

③ Delete all images :-

→ `docker rmi -f $(docker images -q)`