clarusway

DOCKER

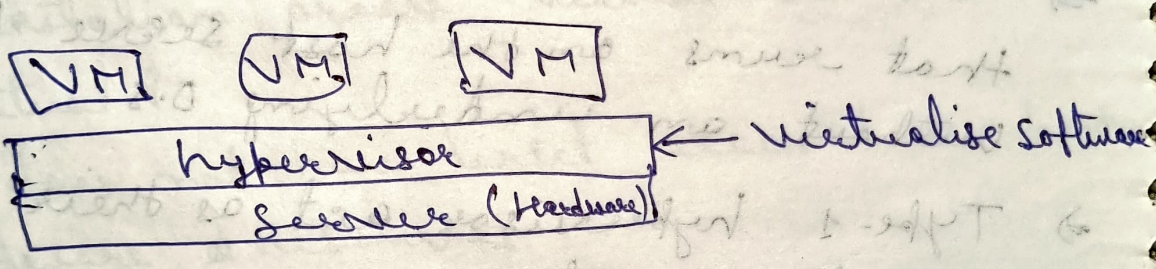
Docker is a tool which create V.M (virtual machine)

Virtualisation

It is the technique of splitting and adding a physical Resource into as many logical Resources as we eg → CPU, Memory.

OR

virtualisation is a technology that transforms hardware into software.



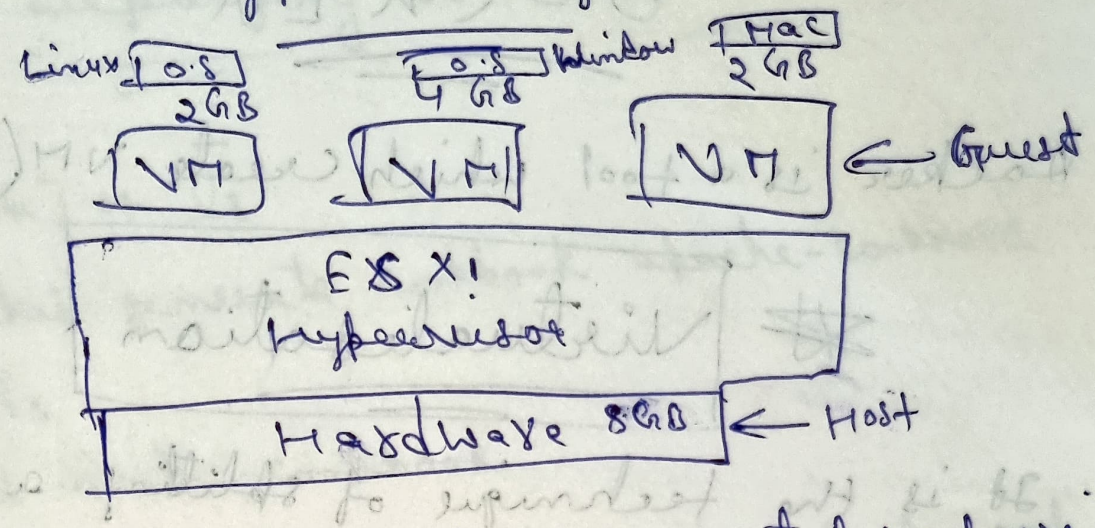
☆ Types of Hypervisor (ESIX)

Hypervisor is a piece of software or firmware that creates and runs virtual machine. A hypervisor sometimes also called a virtual machine manager (VMM)

Types of Hypervisor

- Type 1 → Bare metal OR Native Hypervisor
- Type 2 → Hosted Hypervisor

Type-1 HyperVisor



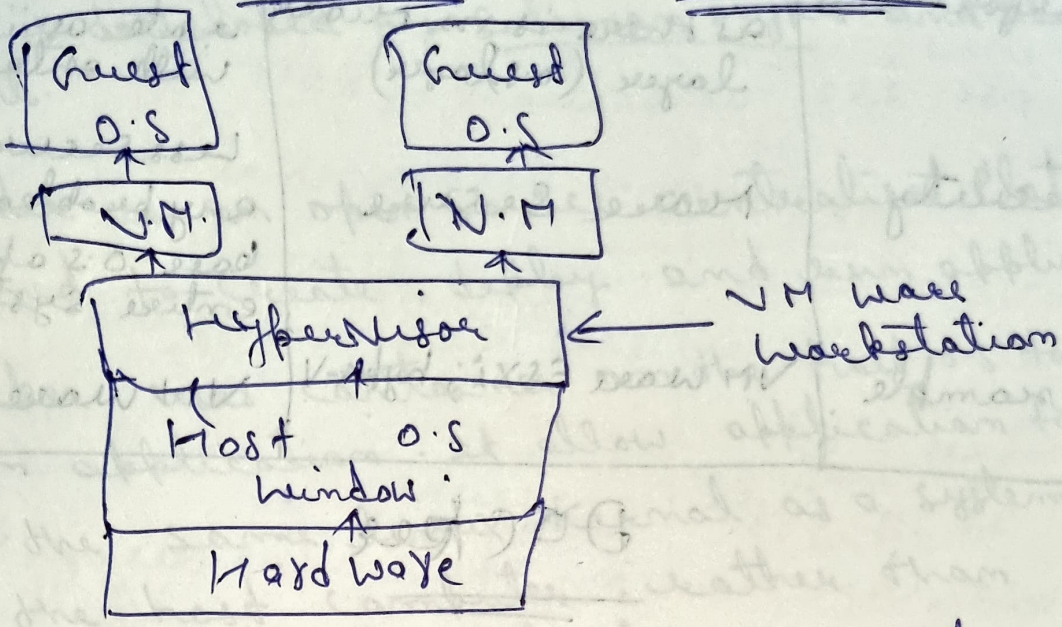
⇒ Type-1 also called Bare metal hypervisor Type-1 hypervisor runs directly on the system hardware. A guest O.S runs on another level above the hypervisor.

⇒ VM ware ESXi is a type-1 hypervisor that runs on the host server hardware without an underlying O.S.

⇒ Type-1 hypervisor act as their own operating system.

⇒ ESXi provides a virtualization layer that abstract the CPU, storage memory and networking Resources of the physical host into multiple virtual machine.

Type - 2 Hypervisor (Hosted type)



⇒ Hypervisor that runs within a conventional O.S environment and the host O.S provides

⇒ Example of type-2 hypervisor are VMware workstation, oracle virtual BOX & Microsoft virtual P.C

⇒ It does not have direct access to the host hardware & Resources

Difference in both type 1 & type 2

Name	Type 1	Type 2
virtualization	Bare metal & native hardware virtualization	Hosted O.S virtualization
operation	Guest O.S & applications run on the hypervisor	Run as an application on the host O.S.
Scalability	Better scalability	not so much because of its Resilience on the underlying OS
System Independance	has direct access to hardware along with virtual machine it host.	are not allowed to directly access the host hardware & its Resources.

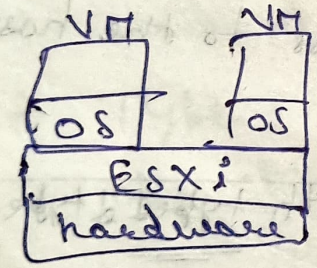
Performance	Higher performance as there is no middle layer (O.S layer)	Comparatively has reduced performance rate as it runs with extra overhead
Security	More Secure	Less secure, bcz as any problem in the base O.S affect the entire system.
Example	VM ware ESXi, hyper-V	VM ware workstation

DOCKER

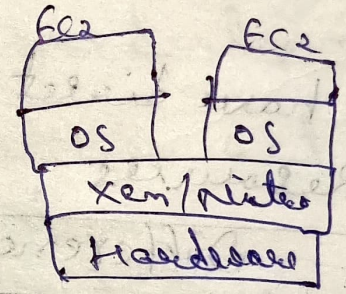
⇒ Docker is a advance version of virtualization

⇒ Docker made container

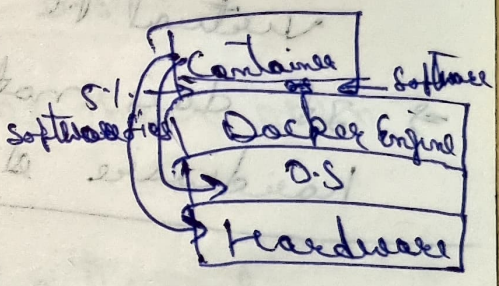
VM ware



AWS EC2



Docker



Docker engine creates container. In container only 5-1. files of O.S files are stored other files are accessed by main O.S of machine & Docker access hardware which we need not taken a fixed storage. According to need of storage it was accessed by hardware of machine. In Docker machine many container are worked but in V.M. can't worked many VM machine

⇒ Docker was first Release in March 2013.
It is developed by Solomon Hykes and Sebastien Pahl.

⇒ Docker is an open source centralised Platform designed to create, deploy and run applications.

⇒ Docker uses container on the host O.S to run application. It allow application to use the same linux Kernal 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 was written in 'go' language.

⇒ Docker is a tool that performs OS level virtualization, also known as 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.

⇒ [Docker is a set of Platform as a service that uses O.S level virtualization whereas] [VM ware uses Hardware level virtualization]

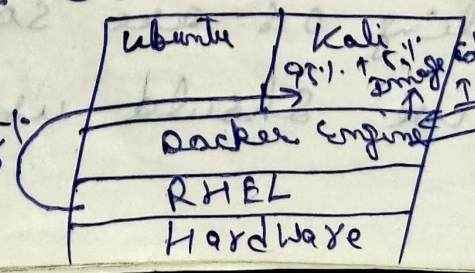


Image are 5% files

⇒ Docker was first Release in March 2013. It is developed by Solomon Hykes and Sebastian Pahl.

⇒ Docker is an open source centralised platform designed to create, deploy and run applications

⇒ Docker uses container on the host o.s to run application. It allow application to use the same linux Kernal as a system on the host computer, rather than creating a whole virtual o.s.

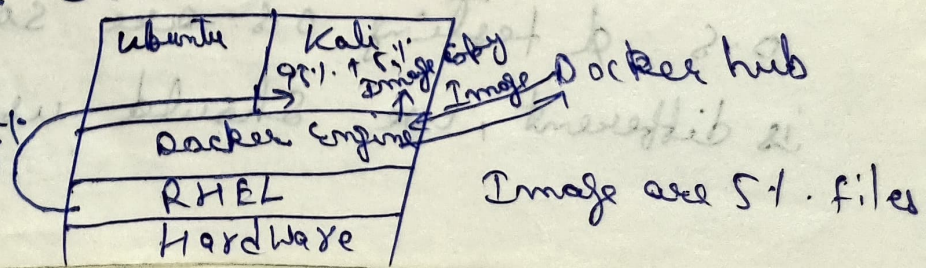
⇒ We can install docker on any o.s but docker engine runs natively on linux distribution.

⇒ Docker was written in 'go' language.

⇒ Docker is a tool that performs os level virtualization, also known as 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.

⇒ [Docker is a set of Platform as a service that uses o.s level virtualization whereas] [VM ware uses hardware level virtualization]



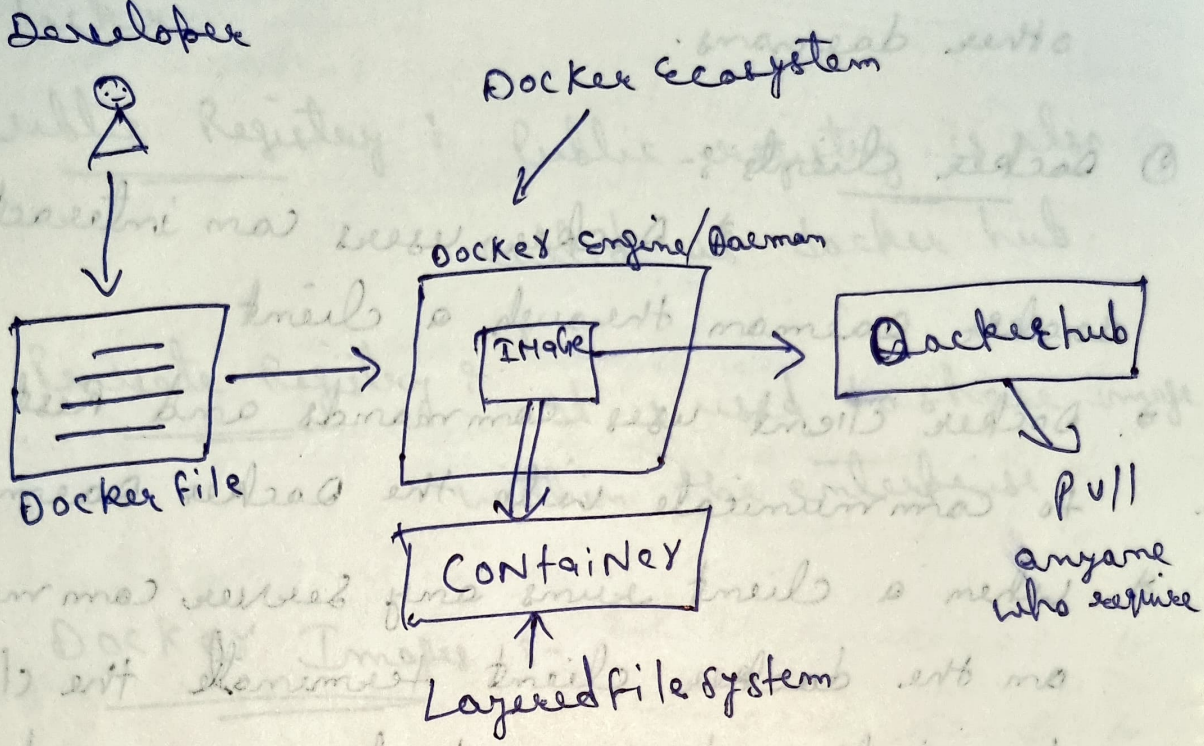
Advantages of Docker

- No less - allocation of RAM.
- C I 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 hardware / virtual H/W or on cloud.
- You can re-use the image.
- It took very less time to create container

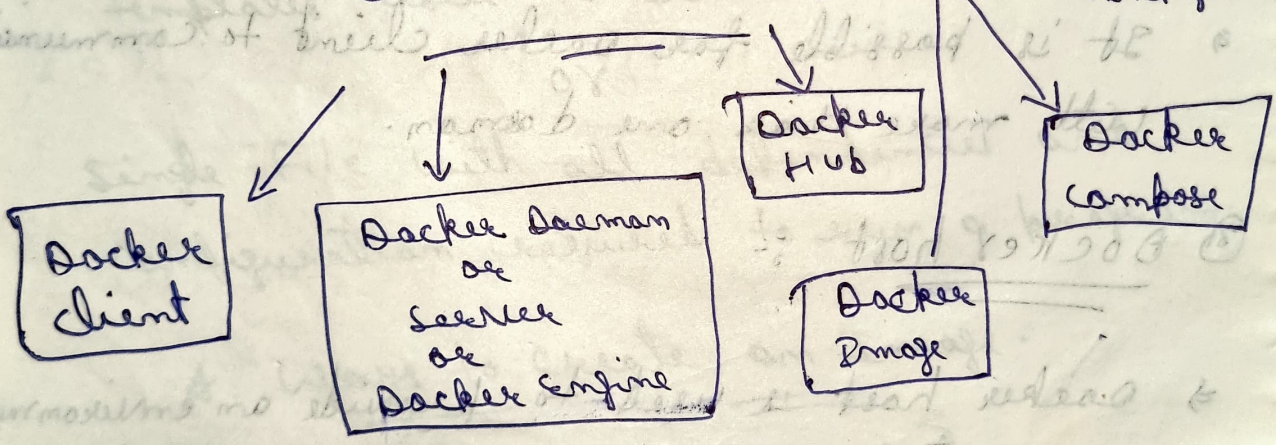
Disadvantage of Docker

- Docker is not a good solution for application that requires rich GUI.
- Difficult to manage large amount of container
- Docker does not provide cross - platform compatibility means if an application is designed to run in a docker container on windows, then it can't run on linux or vice - versa.
- Docker is suitable when the development O.S & testing O.S are same if the O.S is different, we should use V.M.

Architecture of Docker



Docker Ecosystem ⇒ Set of software or Packages



Component of Docker

① **Docker Daemon :-**

- Docker daemon runs on the host O.S.
- It is responsible for running containers to manages docker services.

→ Docker daemon can communicate with other daemons.

(2) Docker clients :-

→ Docker users can interact with docker daemon through a client.

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

→ When a client runs any server command on the docker client terminal, the client terminal sends these docker command 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 application. It contains the docker daemon, images, containers, networks and storages.

(4) Docker Hub / Registry :-

Docker registry manages and stores the docker images.

There are two types of registries in ~~which~~ the docker.

① Public Registry : Public registry is also called as docker hub.

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

⑤ Docker Images :-

⇒ Docker image are the read only binary templates used to create docker containers.

or

single file with all dependencies and configuration required to run a program.

★ Ways to create an Image.

① Take image from docker hub.
② create image from docker file.
③ create image from existing docker containers.

Docker container

Docker version ⇒ 20.10.17

⇒ container hold the entire packages 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.

⇒ Image becomes container when they run on docker engine.

COMMANDS FOR DOCKER

- To see all images present in your local machine

[ ] # Docker images

- To find out images in docker hub

[ ] # Docker search <FileName>

- To download image from dockerhub to local machine

[ ] # Docker pull <ImageName>

- To give name to container

[ ]# Docker run -it [--name &xyz /bin/bash]

Interactive mode — terminate

- To check service is start or not

[ ]# service Docker status

- To start container

[ ]# Docker start <&xyz name of container>

- To go inside container

[ ]# Docker attach < container name >

- To see all containers

[ ]# Docker PS -a

- To see only running containers.

[ ]# Docker PS

- To stop container

[ ]# Docker stop < container Name >

- To delete container

[ ]# Docker rm <container name>

- To Start Docker

[ ]# service Docker Start

Type of Make Docker Image & diff command

- EC2 Instance

*. Now create container from our own image
Therefore, create one container first.

- Docker run -it --name rohit ubuntu /bin /bash
- cd tmp
- Create file inside this "tmp" directory
- touch myfile
- Exit.

*. Now if you want to see the difference b/w the base image & changes on it then.

- Docker diff rohit

O/P
C /root
A /root/.bash_history
C /tmp
A /tmp/myfile

*. Now create image of this container

- Docker commit rohit newfile
- Docker Images

* Now create container from this Image.

→ Docker run -it --name Kapil newfile /bin/bash

root @ cid # ls
cd tmp
tmp #. ls

O/P :- myfile.

Third method to build a Container is By Dockerfile

Docker file :- It is basically a text file. It contains 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 Name.

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 file at docker image side.

EPOSE :- To expose ports such as port 8080 (Jenkins) for tomcat, port 80 (Internet) for nginx etc. "Given a particular port to work".

CMD :- execute commands but during container creation.

ENTRYPOINT :- Similar to CMD, but has higher priority over CMD, first command will be executed by "ENTRYPOINT" only.

ENV :- environment variables {like mohit is replaced to kohli}

ARG :- Argument

LAB WORK {Dockerfile}

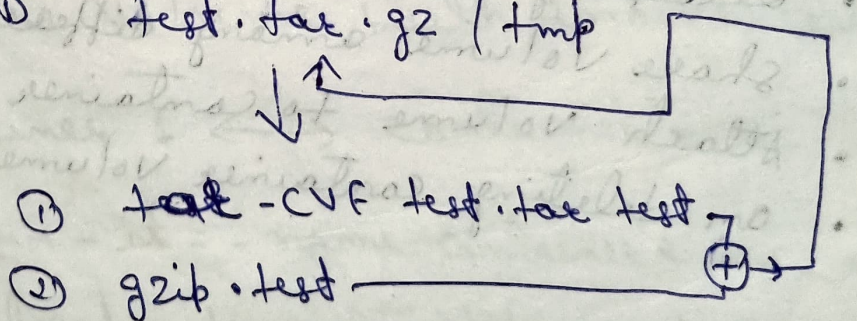
1) create a file named "Dockerfile" only.
2) Add instruction in dockerfile.
3) Build dockerfile to create image.
4) Run image to create container.

Insert with I ← Vi Dockerfile

```
FROM ubuntu
RUN echo "my name is Mohit" > /tmp/myfile
```

- To create image out of dockerfile
  - docker build -t myfile . ← [it shows the current directory]
  - docker ps -a
  - docker Images
- Now create container from the above image
- docker run -it --name Rao myfile /bin/bash
- cat /tmp/myfile

- Vi Dockerfile
  - FROM ubuntu
  - WORKDIR /tmp
  - RUN echo "My name is Rohit" > /tmp/file1
  - ENV myname rohit
  - COPY ~~test~~ testfile /tmp
  - ADD test.tar.gz /tmp



✡ Create Volume In Docker

- Volume is simply a directory inside our container
- Firstly, we have to declare this directory as a volume & 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 access any no. of container.
- volume will not be included when you update an image

① container ⟷ container
② Host ⟷ container

✡ Benifits of Volume

- Decoupling container from storage.
- share volume among different container.
- Attach volume to container.
- on deleting container volume does not delete.

Lab WORK (Volume create in Docker)

★ Creating Volume from Dockerfile

① Create a Dockerfile and write (By vi Dockerfile)

⇒ vi
```
FROM ubuntu
VOLUME ["/myfVolume1"]
```

- Then create image from this Dockerfile

⇒ Docker build -t myimage .

- Now create a container from this image & Run

→ Docker run -it --name container1 myimage /bin/bash.

- Now do ls to see myfVolume1

★ Now share Volume with another container

container 1 ⟷ container 2

- Docker run -it --name container2 --privileged=true --volumes-from container1 ubuntu /bin/bash.

Now after creating container2, myvolume1 is visible whatever you do in one volume, can see from other volume.

- touch /myvolume1/samplefile
- docker start container1
- docker attach container1
- ls /myvolume1

you can see samplefile here
exit

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 more container, and share volume2

→ Docker run -it --name container4 --privileged=true --volumes-from container3 ubuntu /bin/bash

- now you are inside container4 do LS, you can see volume2
- now create one file inside this volume & then check in container3, you can see that file.

☆ Create Volume (Host - container)

- Verify files in Go into /home/ec2-user
- Docker run -it --name hostcont -v /home/ec2-user:/Hotit --Privileged=true ubuntu /bin/bash (container ka andar file jo h)
- cd /Hotit

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

→ touch Hotit (in container)

→ Exit

☆ Now check in EC2 machine, you can see this file

SOME Other COMMANDS

- Docker Volume ls (all volumes created in local M/c)
- Docker volume create <volumename> (To create Volume)
- Docker Volume rm <volume name> (To delete volume)
- Docker Volume Prune (It remove all unuse files)
- Docker Volume inspect <volumename> (To check details related to volume)
- Docker container inspect <containername> (To check details related to container)

HOW TO EXPOSE Docker (container) to Server

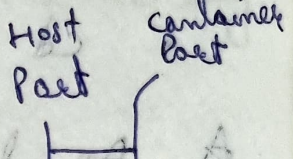
DOCKER PORT EXPOSE

→ Login as ec2-user / sudo su

→ yum update -Y

→ yum install Docker -Y

→ service Docker start



→ Docker run -td --name Mohit -P 80:80 ubuntu

(daemon) ↑ {This command is used to make container only}

→ Docker PS

→ Docker Port rahit (To see which Port expose)

O/P - 80/TCP → 00.0.0/80

→ Docker exec -it rahit /bin/bash (To go inside container)

→ apt-get update (To update our m/c on ubuntu)

→ apt-get install apache2

→ Cd /var/WWW/html (go inside html dir)

→ echo "My Name is rahit" > index.html

→ service apache2 start

→ Docker run -td --name kapil -P 8080:8080 Jenkins

Difference b/w Docker attach & Docker exec?

- Docker `exec` creates a new process in the container's environment while `Docker attach` just connect the standard I/P / O/P of the main process inside the container to corresponding standard I/P / O/P error of current terminal.
- `Docker exec` is specifically for running new things in a already started container, be it a shell or some other process.
- P id → process id / PP id → Parent process id.

D/f B/w expose & Publish a Docker

- Basically you have three options:

1) Neither specify `expose` nor `-P`
2) only specify `expose`
3) Specify `expose` and `-P`

(1) If you specify neither `expose` & `-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 containers are access, so this is good for inter-container communication.

③ 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 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' include expose.

Publish is More Powerfull.

Lec - 30

☆ How to Push docker image in dockerhub

- ecs-user / sudo su
- service docker start
- docker run -it --name rohit ubuntu /bin/bash

☆ Now create some files inside container

☆ Now create image of this container

- Docker commit mohit (image id)

✱ Now create account in hub.docker.com

✱ Now go to EC2 instance

→ Docker login
Enter your username & Password (of dockerhub)

✱ Now give tag to your image.

- Docker tag mohit mohityadav80/Kabil/Project1 ← (new name)
- Docker Push mohityadav80/Project1

✱ Now you can see this image in dockerhub account.

✱ Now create one instance in tokyo region and Pull image from hub.

→ Docker Pull Kabil (Image name)

- Docker run -it --name Rohan Kabil /bin/bash

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)