→ To make jenkins user as root user

```
docker exec -u 0 -it container_name /bin/bash
```

zero

→ To copy all the contents of file into its parent directory i.e one step back

```
cp -R * ../.
```

→ in current directory

one step back

all the files in that directory

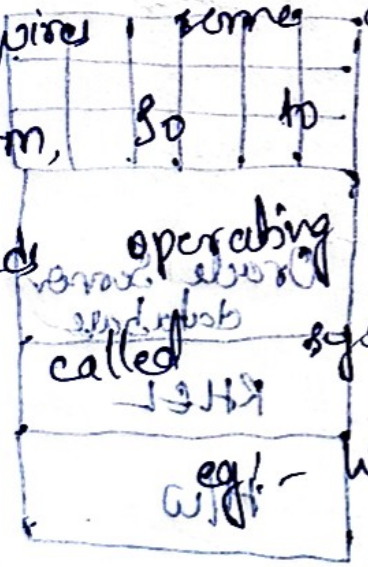
Docker :-

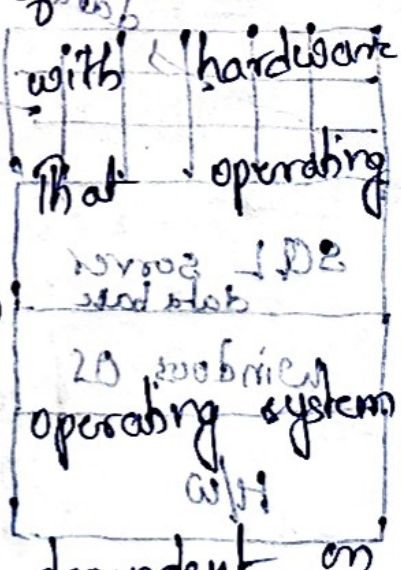
It is a containarization technology

→ Generally we have two softwares, they are (i) application s/w (ii) system s/w.

(i) application s/w :- These are the s/w's we use such as facebook, whatsapp, skype etc

(ii) system s/w :- The application like skype we use directly can't interact with hardware, it requires some amount of RAM, so to interact with hardware, CPU, hardware, it needs operating system. That is called system s/w eg:- Windows operating system.



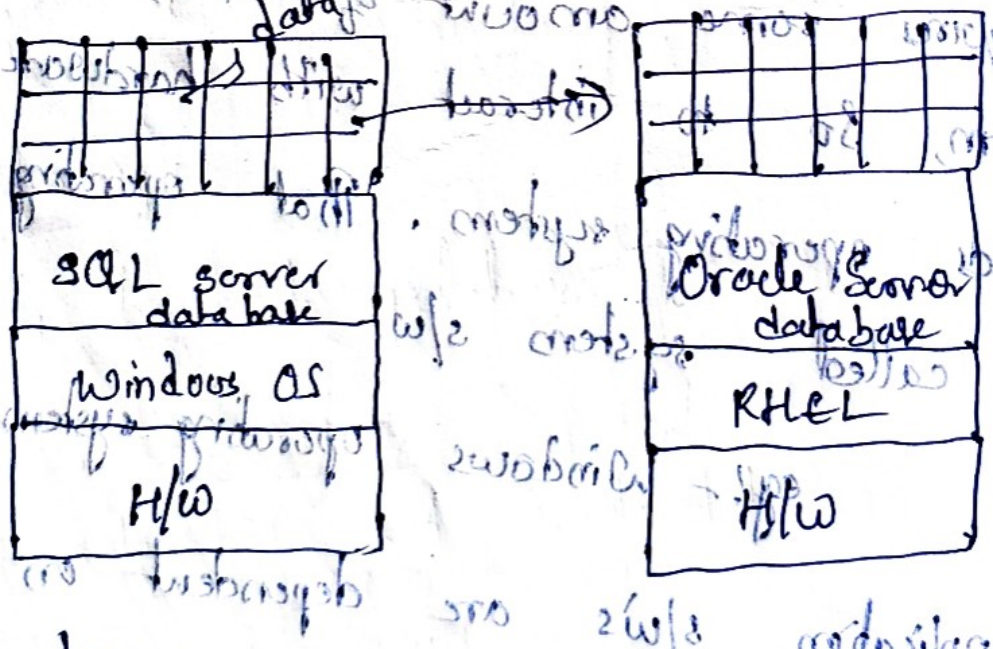


→ Application s/w's are dependent on OS ie system s/w's

→ Scenario:

In a organization, they are using server, on top of that, they are using windows OS, on top of that they are using SQL server database, on top of that they are using data for storage.

→ If they want to transfer that total data into another environment, on which they use linux servers, it creates with expensive.



disadvantages:-

→ Needs another H/W

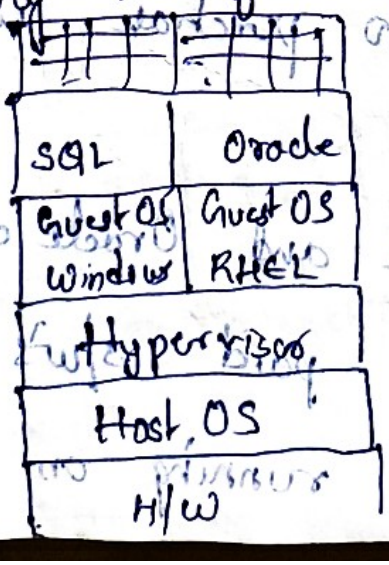
→ RHEL OS
→ Oracle server data base
→ These all factors makes expensive,

Then a technology called Virtualizabon came into picture.

Virtualizabon:-

This is the proces of running multiple OS's parallely on a single piece of H/w.

→ Here we use only one H/w, on top of it we have host OS and on the top of host OS we install an application called as hypervisor, On the hypervisor we can run any no. of OS's as guest OS.



Advantages! —

→ Cost for the H/W is reduced.

Hypervisor!

It is an application S/W, on which we can run multiple OS's as guest OS.

→ The entire cloud is happening on virtualization.

→ The underlining technology for cloud is virtualization

→ AWS cloud works on underlining technology called citizen

Disadvantages! —

→ They have to purchase windows OS and RHEL OS.

→ SQL data base and Oracle data base

→ These are all paid S/W's

→ The application running on the guest OS

have to pass through 'n' no. of layers to access the H/w resources.

→ This will reduce the speed, increases the run time, and becomes slow.

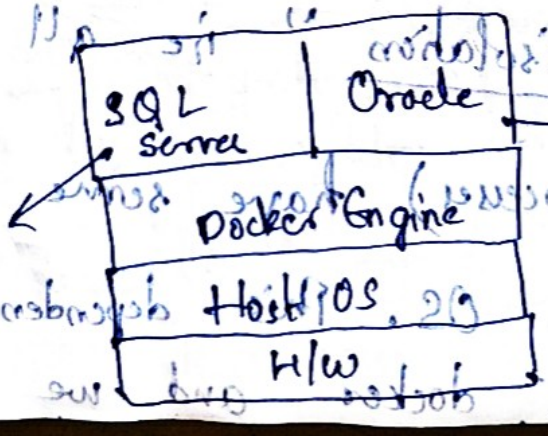
→ In this case docker came into picture.

→ Earlier we are using windows OS for SQL database and RHEL OS for Oracle database.

→ Now the docker has removed that windows and RHEL OS as guest.

→ Now docker is used as "docker Engine" on top we can directly use SQL database and Oracle database.

→ The docker engine has removed the OS dependency.



Containers are the applications which are running on docker engine without the dependency of underlining operating systems this technology we call it as process isolation

Containerization

→ Here we have H/w piece
→ On top of H/w, we have host OS
→ On top of host OS we have docker Engine.
→ On docker engine we can run any no. of applications in the form of containers
→ Docker is a technology for creating these containers.
→ Docker achieve what is commonly called as "process isolation" i.e all the applications (processes) have, some dependency on a specific OS. This dependency is removed by docker and we can run

them on any OS as containers if we have Docker engine installed.

Advantages:-

→ These containers pass through less no. of layers to access the H/w resources.

→ The organizations need not spend money on purchasing licenses of different OS's to maintain various applications.

→ Docker can be used at the stages of S/w development life cycle.

Build --> ship --> Run

(*) Docker is available in Community edition (free) and Enterprise edition (paid)

Setup of Docker on Windows

① Download docker desktop from

https://www.docker.com/products/docker-desktop

② install it, it is accessed from Power shell

Build → ship → Run

Build is dev environment, where code is developed.

ship → it is a environment where packaging and testing is done.

Run → production environment.

→ These all stages can be dockerized.

→ Docker is very fast, within matter of seconds docker can install any application or OS and it can remove that installed OS or any application within seconds.

Install docker, available in get.docker.com

```
curl -fsSL https://get.docker.com -o install-
                                    docker.sh
```

```
sudo sh install-docker.sh
```

Docker image :- (bin/libs)

By definition it is a combination of binaries and libraries which are necessary for a software application.

Docker container :-

Once the image came into running condition it is called as container.

→ The original setup file is known as image

→ The installed application in docker terminal is called container

→ We can create n no. of containers with one image.

→ The machine which is used to run docker commands is called docker host

→ We have docker client and docker deamon

Docker client :-

It takes all the docker commands and passes through the docker deamon.

Docker deamon :-

It takes those commands and do processing according to the command.

→ There two proces run in background

→ The docker deamon either route those commands to download images or to create containers and some commands made to works for registry.

→ The registry is classified in two types

(i) Public registry :- It can be accessed by all the people.

eg :- hub.docker.com

(ii) Private registry :- Only our organisation can access

eg :- Our own registry

Image:-

A docker image is a combination of bin/libs that are necessary for a s/w application to work. Initially all the s/w's of docker are available in the form of docker images.

→ A running instance of an image is called as a container

Docker host:-

The server where docker is installed is called docker host.

Docker registry:-

This is the cloud site of docker where docker images are stored.

Working on docker images :-

1. To pull a docker image.

```
docker pull image_name
```

2. To search for a docker image.

```
docker search image_name
```

3. To upload an image into docker hub

```
docker push image_name
```

4. To see the list of images that are downloaded

```
docker images
```

```
docker image ls
```

5. To get a detailed information about docker image.

```
docker image inspect image_name/image_id
```

⑥. To delete a image that is not linked to any container

```
docker rmi image_name / image_id
```

⑦. To delete a image that is linked to a container

```
docker rmi -f image_name / image_id
```

⑧. To save the docker image as a tar file.

```
docker save image_name
```

⑨. To untar this tar file and get image

```
docker load tarfile_name
```

⑩. To delete all images

```
docker system prune -af
```

(11). To create a docker image from a docker file.

```
docker build -t image_name.
```

(12). To create a image from a customised container.

```
docker commit container_id/container_name image_name
```

→ To rename a image

```
docker tag old_image_name newimage_name
```

Working on docker containers :-

(13). To see the list of running containers

```
docker container ls
```

(14) To see the list of all containers (running and stopped)

```
docker ps -a
```

(15). To start a container

```
docker start container_name/container_id
```

(16). To stop a container

```
docker stop container_name/container_id
```

(17). To restart a container

```
docker restart container_name/container_id
```

→ To restart after 10 seconds

```
docker restart -t 10 container_name/container_id
```

(18). To delete a stopped container

```
docker rm container_name/container_id
```

(19). To delete a running container

```
docker rm -f container_name/container_id
```

(20). To stop all running containers:

```
docker stop $(docker ps -aq)
```

(21). To delete all stopped containers:

```
docker rm $(docker ps -aq)
```

(22). To delete all running and stopped containers.

```
docker rm -f $(docker ps -aq)
```

(23). To get detailed info about a container

```
docker inspect container_name/container_id
```

(24). To see the logs generated by a container

```
docker logs container_name/container_id
```

(25). To create a docker container

```
docker run image_name/image_id
```

run command options:-

--name : used to give a name to the container.

--restart: Used to keep the container in runing condition.

-d : used to run the container in detached mode in background.

-it : used to open interactive terminal in the container

-e : used to pass environment variables to the container.

-v : used to attach an external device or folder as a volume.

--volumes-from : Used to share volume between multiple containers.

-p : used for port mapping. It will link the container port with host port.

eg:- -p 8080:80 ; here 8080 is host port (external port) and 80 is container port (internal port)

-P.: used for automatic port mapping where the container port is mapped with some host port that is greater than 30000.

--link: used to create a link between multiple containers to create a microservice architecture.

--network: used to start a container on a specified network.

-rm: used to delete a container on exit.

-m: used to specify the upper limit on the amount of memory that a container can use

-c: used to specify the upper limit on the amount of CPU a container can use

-ip: used to assign an ip to the container

(26) To see the ports used by a container:

```
docker port container_name/container_id
```

(27) To run any process in a container from outside the container.

```
docker exec -it container_name/container_id
                process_name
```

eg:- To run the bash process in a container

```
docker exec -it container_name/container_id bash
```

(28) To come out of a container without exit

```
ctrl+p , ctrl+q
```

(29) To go back into a container from where the interactive terminal is running.

```
docker attach container_name/container_id
```

30. To see the processes running in a container

```
docker container container_name/container_id top
```

Working on docker networks

31. To see the list of docker networks

```
docker network ls
```

32. To create a docker network

```
docker network create --driver network_type network_name
```

33. To get detailed info about a network

```
docker network inspect network_name/network_id
```

34. To delete a docker network

```
docker network rm network_name/network_id
```

(35) To connect a running container to a network

```
docker network connect network_name/network_id
container_name/contained_id
```

(36). To disconnect a running container to a network.

```
docker network disconnect network_name/network_id
container_name/container_id
```

Working on docker volumes

(37). To see the list of docker volumes

```
docker volume ls.
```

(38). To create a docker volume

```
docker volume create volume_name
```

(39). To get detailed info about a volume

```
docker volume inspect volume_name/volume_id
```

(40) To delete a volume

```
docker volume rm volume_name/volume_id
```

→ nginx is a reverse proxy server, used to control the networks in the organisation, To restrict the some of the websites in organisation.

→ We can access any webapplication through port mapping.

→ We can change host port of a container

→ Container port is predefined

→ To change

→ To do port mapping of nginx

eg:

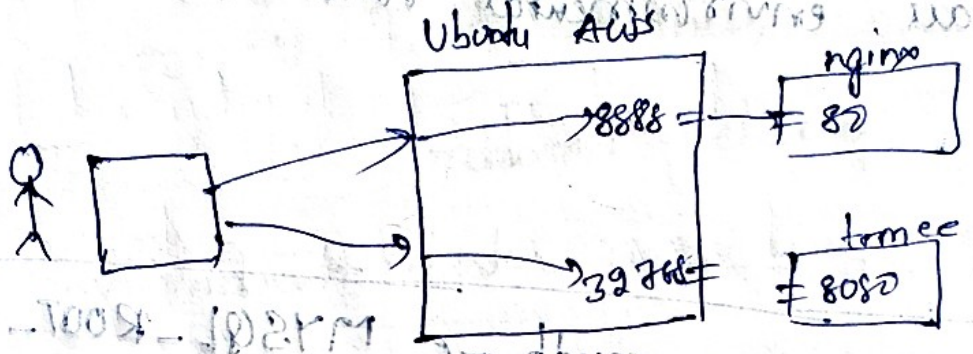
```
docker run --name webserver -d -p 8888:80 nginx
```

To access the nginx from browser give

```
public_ip_address of host : 8888
```

→ For automate port mapping used -P
it will automatically assign the port no
which is greater than 30,000 number

eg: `docker run --name cappserver -d -P tomee`



→ We can accen the any webapplicabon through

`public-ip of host : port no`

through public ip, I can reach upto the
Ubuntu server which is set on cloud,
from their to accen the webapplicabon
we use portno which we assigned
to the particular container.

→ To launch any OS, it will be launched via terminal, we use -it

eg:- To install ubuntu OS

```
docker run --name u1 -it ubuntu
```

→ To pass environmental variables we use -e

eg:-

```
docker run --name mydb -e MYSQL_ROOT_PASSWORD = password -d mysql
```

↓

This data will be available in docker hub

→ To go inside a running container

eg:

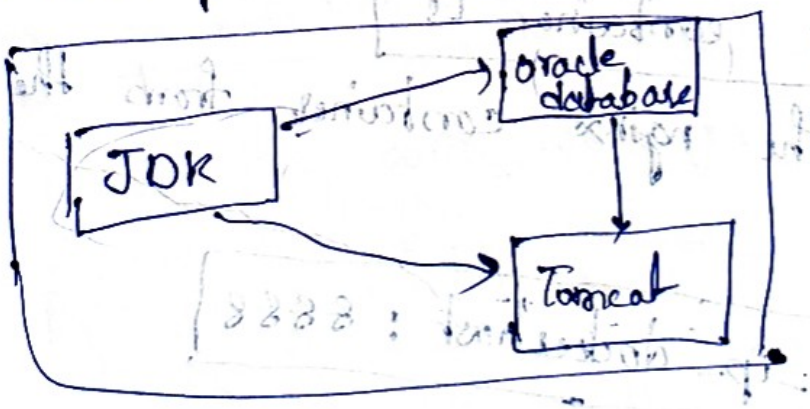
```
docker exec -it mydb bash
```

Multi container Architecture or Micro services architecture:

→ To create a environment in an organisation eg:- development environment, or testing environment, and link those environment with another;

→ To establish network between different environments, we use multi container architecture

eg:- Development environment



1. --link option
2. docker compose
3. docker networks

We also setup using:

(4) Ansible playbooks

(5) Python scripts

Scenario:-

(*) Create an nginx container in detached mode and name it as webserver, also perform port mapping.

→ `docker run -it --name webserver -p 8888:80 -d nginx`

To check if the nginx container is running

`docker container ls`

To access the nginx container from the level of browser

`public_ip_dockerhost : 8888`

Scenario:-

Start jenkins container in detached mode and also perform port mapping.

```
docker run --name myjenkins -d -p 9999:8080
jenkins/jenkins
```

To see the ports used by the above server

```
docker port myjenkins
```

To accey it from the browser

```
public_ip_of_host : port_no_from_above_comad
```

Scenario! - Start tomcat and do automatic port mapping

```
docker run --name appserver -d -P tomcat
```

To see the ports used by the above server

```
docker port appserver
```

To accey it from the browser

```
public_ip_host : port_no_from_above_command
```

Scenario 1:-

start centos as a container and launch interactive terminal in it.

```
docker run --name mycentos -it centos
```

To come out of the container

```
exit
```

Scenario 2:-

Create a mysql container and login as root user and create a sql table

→ Create a mysql container

```
docker run --name mydb -d -e MYSQL_ROOT_PASSWORD='sudarshan' mysql:5
```

→ docker container ls

→ To go into the bash shell of the container.

```
docker exec -it mydb bash
```

→ To login into the database

```
mysql -u root -p
password:
```

→ To see the list of databases

```
show databases;
```

→ To move into any of the database

```
use database_name;
```

eg:- use sys;

→ To create emp and dept tables there is no direct way

→ move into library of the data base

→ open emp and dept tables for mysql in google, and copy those commands and paste in bash shell.

→ To see the data of the tables

```
select * from emp;
select * from dept;
```

Scenario1:

Create two busybox containers c1 and c2 and link them

→ To create a c1 container

```
docker run --name c1 -it busybox
```

→ Now create c2 container and link with c1

```
docker run --name c2 -it --link c1:my busybox busybox
```

→ Check if c2 is linked or not.

```
ping c1
```

Scenario2:

Setup wordpress and link it with mysql container.

→ Create a mysql container

```
docker run --name mydb -d -e MYSQL_ROOT_
PASSWORD=sudarshan mysql:5
```

→ Now create wordpress and link with mysql database

```
docker run --name mywordpress -d -p 8888:80
--link mydb:mysql wordpress
```

→ To access it from browser

public_ip_host: port_no - wordpress in above command (8888)

→ To check wordpress is linked or not

```
docker inspect wordpress
```

and search for the "Links" section

Scenario:

Setup CI-CD environment where jenkins is installed in a container and linked with a tomcat container for QA server and Prod Server

① Create jenkins container

```
docker run --name myjenkins -d -p
5050:8080 jenkins/jenkins
```

② To access jenkins from the browser

host_public_ip:port_no_in_above command

③ Create tomcat container for QA environment link with jenkins

```
docker run --name qaserver -d -p 6060:8080
--link myjenkins:jenkins tomee
```

④ Create tomcat container for prod server and link with jenkins

```
docker run --name prodserver -d -p 7070:8080
--link myjenkins:jenkins tomee
```

⑤ docker container ls

⑥ Access all those from browser.

Scenario!-

Create a postgres container and link with adminer container.

Note!- adminer is a client application of databases.

(1) Create a postgres container.

```
docker run --name mydb -d -e POSTGRES_USER=sudarshan -e POSTGRES_DB=mydb postgres
```

(2) Create a adminer and link it with mydb

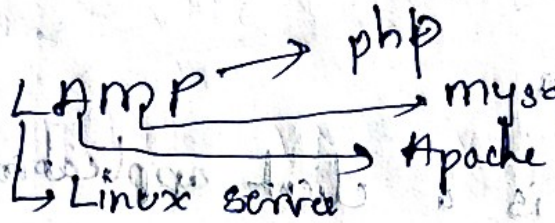
```
docker run --name myadminer -d -p 8888:8080 --link mydb:postgres adminer
```

(3) To access it from browser

```
public-ip-host:8888
```

Scenario:-

setup LAMP architecture



① Create mysql container

```
docker run --name mydb -d -e MYSQL_ROOT_
PASSWORD=sudarshan mysql.
```

② Create an apache container and link with mysql container

```
docker run --name apache -d -p 9999:80
--link mydb:mysql httpd
```

③ Creat a php container and link with mysql and apache

```
docker run --name php -d --link mydb:mysql
--link apache:httpd php:7.2-apache
```

④ To check if php container is linked with apache and mysql

```
docker inspect php
```

Scenario ** Interview

Create a testing environment where a selenium hub container is linked with 2 node containers one with chrome and other with firefox installed.

① Create a selenium hub container

```
docker run --name hub -d -p 4444:4444
selenium/hub
```

② Create a container with chrome installed on it

```
docker run --name chrome -d -p 5901:5900
--link hub:selenium selenium/node-chrome-
debug
```

③ Create another container with firefox installed on it

```
docker run --name firefox -d -p 5902:5900
--link hub:selenium selenium/node-firefox-
debug
```

④ The above 2 containers are GUI ubuntu

containers and we can access their GUI using VNC viewer

(a) Install VNC viewer from

https://www.realvnc.com/en/connect/download/viewer/

(b) Open vnc viewer → public ip of docker host :: 5901 or 5902

Click on continue → enter password: secret

Docker compose :-

→ The disadvantage of "link" option is it is depricated and the same individual command have to be given multiple times to setup similar architecture.

→ To avoid this we can use docker compose

→ Docker compose uses yml files to setup the multi container architecture and these

files can be reused any number of times

→ To create a docker file use

```
vim docker-compose.yml
```

Format:-

```
version: 3.8
services:
    container_name:
        image_name:
        ports:
        environment:
            ↳ if it is present in
              a concerned architecture
    ...     links;
```

→ To setup the containers from the above file

```
docker-compose up -d
```

→ To stop and delete the containers

```
docker-compose down
```

→ To start the container

```
docker-compose start
```

→ To stop all the containers of the docker compose file

```
docker-compose stop
```

(*) Setup a testing environment where selenium hub is linked with 2 node containes one with chrome and firefox

```
---
version: '3.8'
services:
  hub:
    image: selenium/hub
    ports:
      - 5000:4444
```

```
---
version: '3.8'
services:
  hub:
    image: selenium/hub
    ports:
      - 5000:4444
```

```
  chrome:
    image: selenium/node-chrome-debug
    ports:
      - 5901:5900
    links:
      - hub:selenium

  firefox:
    image: selenium/node-firefox-debug
    ports:
      - 5902:5900
    links:
      - hub:selenium
```

...

* Setup CI-CD environment where jenkins container is linked with QA and Prod Server.

```
---
version: '3.8'
services:
  jenkins:
    image: jenkins/jenkins
    ports:
      - 5050:8080
  qaserver:
```

```
    image: tomee
    ports:
      - 6060:8080
    Links:
      - jenkins:jenkins/jenkins
  prodserver:
    image: tomee
    ports:
      - 7070:8080
    Links:
      - jenkins:jenkins/jenkins
```

...

(*) setup a postgres database and link it with adminer.

```
---
version: '3.8'
services:
  mydb:
    image: postgres
    environment:
      POSTGRES_PASSWORD: sudarshan
  myadminer:
    image: adminer
```

```
    ports :
      - 8180 : 8080
    links :
      - mydb : postgres
```

(*) Setup LAMP architecture

```
---
version : '3.8'
services :
  mydb :
    image : mysql
    environment :
      MYSQL_ROOT_PASSWORD : sudarsha

  apache :
    image : httpd
    ports :
      - 8282 : 80
    links :
      - mydb : mysql

  php :
    image : php : 7.2-apache
    links :
      - mydb : mysql
      - apache : httpd
```

Docker volumes?-

→ Containers are atformral and data should be persistence, atformral means temporary, persistence means should remain the data even the container is deleted.

→ Volumes have backup mechanism in docker.

① Simple docker volumes

It is a simple way of preserving the data even though the container is deleted.

→ Now create a folder in host.

```
mkdir /data
```

→ Now create a container and mount that folder in the container.

```
docker run --name myngingx -d -p
-v /data:ngingx
```

→ The folder created on host machine will also be available in the container

→ Now if you delete the container, the folder will be present

→ To see the folder after deleting the container,

```
docker inspect container_name
```

In that search for the mount, there you can find the path of the mount file, copy that path and cd path of the source

```
cd path of the source
```

→ Now you can see the folder and files in that folder

→ Though the folder is part of the container, but actually it is a part of the host machine.

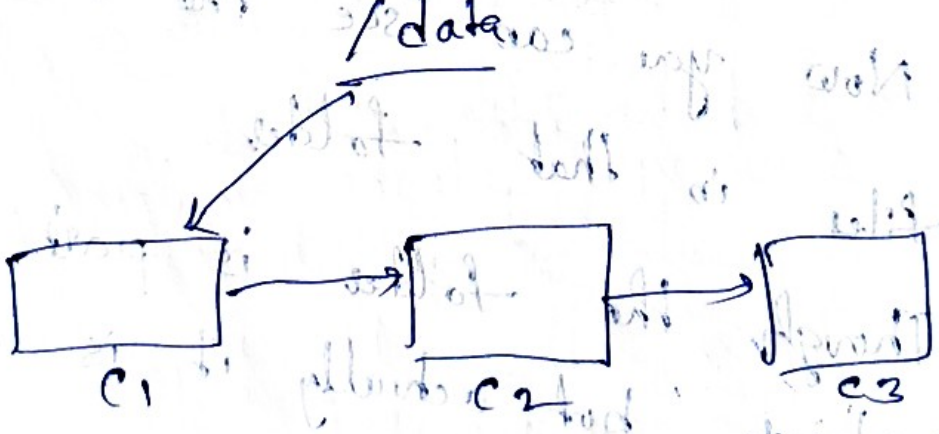
② Sharable docker volumes.

→ The volume which is created on host machine, it is shared between multiple containers.

eg:-

Scenario:-

✱ I will create a /data folder on host machine, and I will mount on the centos container at the time of creating and folder in the first container is shared with second container, and the second container's folder is shared with third container.



→ C2 container should use the volume of C1

→ C_3 should use the volume of C2

→ first create a folder /data.

```
mkdir /data
```

→ Now create a centos image as container.

```
docker run --name c1 -it -v /data
                                centos
```

→ Now share the same volume of /data folder which is used by c1.

```
docker run --name c2 -it --volumes-from
c1 centos
```

→ Now create c3 container and share the volume of c2

```
docker run --name c3 -it --volumes-from
c2 centos
```

→ In all the container, the data created

in one container is seen in all the containers.

→ To see the files navigate to mount section, copy the source path, even if the container is deleted.

(3) Docker volume containers

→ These are bi-directional, i.e the files created on the host machine are available inside the container and vice versa.

→ First create a volume on the host machine

```
docker volume create myvolume
```

→ Now inspect where the volume is located.

```
docker volume inspect myvolume
```

→ search for mount point, copy that path and see the files create some files or put some data in that location

→ Now create a container

```
docker run --name U1 -it -v myvolume:/tmp ubuntu
```

myvolume will be available in the /tmp folder → or give any foldername

→ Even though the container is deleted it will be available

```
docker volume ls
```

Scenario:-

→ In my organisation we created a tomcat container, in which it has many files and folders, In tomcat I have tomcat-users.xml file

and we want to modify that file in 100 tomcat containers,

→ In this case I created a volume in my host machine and mounted it on 100 tomcat containers, we were trying to change tomcat-users.xml file in conf folder.

→ First create a volume

```
docker volume create myvolume
```

→ Now copy the path of the volume

```
docker inspect myvolume
```

copy the path and navigate to that location, and there you create a file tomcat-users.xml

```
cat > tomcat-users.xml
```

→ Here add the data in xml format

→ Now create a tomcat container and mount it in the /tmp folder

```
docker run --name t1 -d -P -v
myvolume:/tmp tomee
```

→ To go into a container

```
docker exec -it t1 bash
```

Here you will see the myvolume in /tmp folder.

→ Now create another tomcat containd and mount that volume, in this way mounted in 100 containers.

→ To delete all the volumes

```
docker volume prune -f
```

→ To take the backup of the softwares which are created in the container, or the softwares which are present are backedup for this customised images comes into picture.

→ Now we create our own images and push them into docker hub.

→ This can be done via docker commit and docker file

→ dockerfile is much more efficient for creating customised images.

→ All the images on the dockerhub has a background file, that file is called dockerfile

Dockerfile → docker image → docker container

Scenario!

① Create a ubuntu container and install git, maven in the container. Now delete that container but you should not loss the softwares installed in the container.

→ whenever a container is created, and some softwares are installed in it, if you delete that container whatever the softwares present in it are deleted. So to avoid that we are going with customised images.

→ First create a container and put some data or install some softwares in the container.

→ Then convert that container into image and delete the container.

→ Now whenever you want everything to be present, create container out of your

customised image.

eg:-

→ First create a ubuntu container

```
docker run --name u1 -it ubuntu
```

→ Now go inside the container

```
docker exec -it u1 bash
```

→ Now install git and maven

```
apt-get update
```

```
apt-get install -y git maven
```

→ Now exit from the container and create a customised image for above container

```
docker commit u1 myubuntu
```

→ Then delete container, image will be present and you can create same container from

that image.

Dockerfile :-

→ It is a simple text file, in that file we give certain instructions, with the help of those instructions we create customised docker images

Keywords in dockerfile :-

1. FROM
2. MAINTAINER
3. RUN
4. COPY
5. ADD
6. EXPOSE
7. VOLUME
8. USER
9. WORKDIR
10. ENV
11. ARG
12. SHELL
13. LABEL
14. CMD
15. ENTRYPOINT

① FROM :-

It represents the base image on which you are doing customisations.

→ Take one base image on top of the base image we can do our own customi-sations.

② MAINTAINER :-

It represents, the name of the organisation/person who created the image.

eg:- If it is created by IBM company

MAINTAINER IBM organisation

(or)

MAINTAINER sudarshan

→ Its is just like signature.

③ RUN :-

It is used to run the Linux command at the time of creating image.

eg:-

RUN apt-get update

RUN apt-get install -y git

④ COPY :-

It is used to copy a file which is present in host machine into container

→ ADD also used for this purpose.

⑤ ADD :-

It is used to copy a file which is present on the remote server into a customised image.

⑥ EXPOSE :-

It is used to define the container port too in customised image.

→ We can assign both container and host port

eg:- 8080 : 80 → Container port

host port

(7) VOLUME :-

If you use volume for the customised image you can mount a volume. ~~without giving~~

→ If you create container, by default a volume is mounted without giving -v is the command.

eg:- VOLUME /data

(8) USER :-

You can add user to the container.

→ By default root will be the user, he has all permissions,

→ For newly created container, user will be root most of the times but for some there is other users.

eg:- For jenkins container user is jenkins, has no powers to install any softwares, then change the user

jenkins to root in our customised image. then all the powers are given to the container user.

(9) WORKDIR :-

when you create a container by default there will be a folder, it will be controlled by WORKDIR.

eg:- go inside a container, then type pwd it shows the folder.

(10) ENV :-

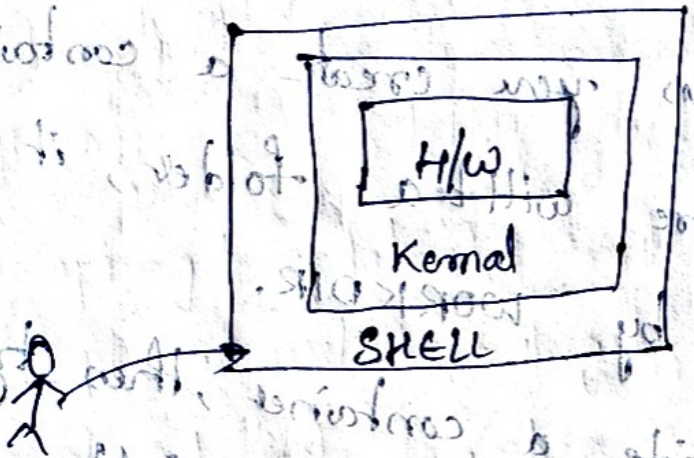
To pass any environmental variables we use ENV

(11) ARG :-

This is used to pass some arguments when the time of creating container, to pass some third party values

(12). SHELL :-

It is an connection between you and underling operating system.



You passing the linux comands to the SHELL, it takes those comands and pass to the kernal, in linux we have shells

eg:- born shell

korn shell

bash shell.

(13). LABEL :-

To store meta data about the image, LABEL is used.

(14) CMD :-

when you create a container, every container triggers one comand which is known as default process of the containers.

→ The container will be in running condition as long as the default process runs.

→ either you can use CMD or ENTRYPOINT.

→ but there will be slight difference

(15) ENTRYPOINT :-

It is also same as CMD.

→ To create a dockerfile

```
vim dockerfile
```

To create a image from the dockerfile

```
docker build -t image_name .
```

eg:- Create a ubuntu image, in that install git and tree softwares.

vim dockerfile

```
FROM ubuntu
MAINTAINER intelliqit
RUN apt-get update
RUN apt-get install -y git
RUN apt-get install -y tree
```

→ Now `docker build -t myubuntu .` ,

image is created.

→ Dockerfiles are in bytes (because it is a simple textfile, we can save the memory), Containers and images have some amount of storage space is more compared to dockerfile.

→ After doing the work you may delete

containers and images. but don't delete dockerfile.

Advantages of dockerfile

→ Its size is less because it is text file

→ You can implement version controlling.

create docker file and upload into the repository, all the team members will clone it and they make changes and upload. but we can find error and at later versions

→ docker

Interview

(*) what are the best practise you followed while creating docker images

A:- Cache busting

(*) what is the difference between COPY and ADD, CMD and ENTRYPOINT

→ For different dockerfiles, eg:- dockerfile1

```
docker build -f dockerfile1 -t image_name .
```

Interview

Cache :-

when a dockerfile is executed, it installs everything which are mentioned in the dockerfile, lets say apt-get update is executed at the time of image creation.

→ All those commands and installed softwares are stored in cache memory of docker.

→ Next time when you execute the same file the apt-get update will not be executed along with other commands will not be executed. because they were executed and stored in cache.

disadvantage :-

→ lets say the apt-get update is not executed, in time being the ubuntu

community who is ~~working~~ maintaining the remote repository has updated, added latest version.

→ Now when you run dockerfile, only older commands ~~got~~ shown on the screen that they are in cache, they got executed later time.

→ This will lose the latest versions of the ∞ softwares

→ So in this case cache busting has came into picture.

→ So when you give cache busting command it will refresh the docker file and executes from starting.

Cache busting :-

→ It is the proces where you can clearly tell you don't want to take instructions from the cache and execute freshly from the starting.

docker build --no-cache -t image_name.

Scenario: This will be used when you are modifying dockerfile after huge gap.

→ In dockerfile we can't use loops, if conditions etc

(*) Installation of docker on a container.

→ We have two commands for this

(i) Curl -fsSL https://get.docker.com -o install-docker.sh.

when you trigger the above command it downloads the shell script into the host machine.

→ Now make a dockerfile to install docker on a container

vim dockerfile

```
FROM ubuntu
MAINTAINER intelliqit
COPY install-docker.sh /
RUN sh install-docker.sh
```

→ COPY command copies the shell script of docker into container and RUN command will install docker in the container.

(*) Now I installed docker in a container. If I am executing docker commands are not running.

Reason:- It will run when you use dind

eg:-

```
docker run --name d1 --privileged docker:dind-rootless
```

(name → container-name)

```
docker exec -it d1 docker-entrypoint.sh sh
```

(d1 → container-name)

⊛ Installation of ansible

vim dockerfile

```
FROM ubuntu
MAINTAINER intellipaat
RUN apt-get update
RUN apt-get install -y software-properties-common
RUN apt-get install -y ansible
```

USER :-

when you create a container, you use the user, for it, sometimes it may be root and other user too.

→ I can make my own user by USER command in the dockerfile.

eg:-

→ Create a jenkins container

→ Now go inside the jenkins see the who is user

whoami

→ Now try to execute apt-get update, it won't allow you to do so, because the user jenkins don't have permissions

→ The root only have all permissions.

→ So to make jenkins user as root, we need USER module

→ If root becomes the user of jenkins, then you can do anything.

→ To go into the root, we need password, because we don't know password we can't go inside the root. In this case USER module comes into picture.

vim dockerfile

```
FROM jenkins/jenkins
MAINTAINER intelligit
USER root
```

→ Now the user is changed as root user, now you can do installations.

(*) For security purpose, I will create a jenkins containa, with the root permission I will install git and maven and make user as jenkins again they nobody will modify;

vim dockerfile

```
FROM jenkins/jenkins
MAINTAINER intelligit
USER root
RUN apt-get update
RUN apt-get install -y git maven
USER jenkins
```

EXPOSE:-

```
FROM nginx
MAINTAINER intelligit
EXPOSE 90
```

→ nginx will run on 80 port by default.

→ Now I changed it to 90 with EXPOSE module

→ To make jenkins user as root user

docker exec -u 0 -it containername /bin/bash

Interview ★★★

(*) Difference between CMD and ENTRYPOINT

ENTRYPOINT :-

→ when we create linux based operating system, we give -it in the command itself, then it directly takes into the bash shell, where as in tomcat, nginx etc we don't give -it in the run command, after creating these containers we give "docker exec -it" command. then it will take into container because for linux based containers entrypoint is bash where as for others entrypoint

is another.

→ For linux based containers the default process is bash shell where as for others it is not bash.

→ With the help of entrypoint we can change ubuntu os nginx, nginx os ubuntu etc.

→ We can change the behaviour of the container.

→ With the help of entrypoint we can customize the software application

eg:- If the team members created a software and tested on ubuntu server

→ They wanted me to create a dockerfile

→ Then we can create a dockerfile, with the entrypoint we can customise the software as another container.

* The moment when you exit the default process of ubuntu container goes into exited state. where as other containers default process is not shell.

→ If we take default process of one container and specify in another container's entry-point, then we can change the behaviour of that container.

eg:- Take the entrypoint of tomcat and specify in the entrypoint of ubuntu then ubuntu will behave as tomcat

Scenario!

* Create jenkins container with dockerfile.

Vim dockerfile

```
FROM ubuntu
MAINTAINER sudarshan
RUN apt-get update
RUN apt-get install -y openjdk-11-jdk
```

ADD past the url of jenkins download

ENTRYPOINT ["java", "-jar", "jenkins.war"]

Scenario:

→ In a organisation, the team of developers created a software eg:- abc. and they tested on linux based servers, they ensure that it is working, they wanted a devops engineer to create a docker image

→ Then we use base image as linux

FROM ubuntu

RUN { whatever s/w's are necessary use RUN }

ADD/COPY { whatever conf files are necessary use ADD/COPY }

ENTRYPOINT [what command they used as running that software.]

Then we create a docker image of their software.

→ echo $PATH

It gives the path of the folder where all the softwares are download.

→ eg:- to see where the nginx is downloaded

→ which nginx → it will show where the nginx is download

→ In the place of ENTRYPOINT we can use CMD it will work, but a minor difference in CMD

CMD :-

→ In the dockerfile if the default process is defined by CMD command, from the command line when you are creating a container, you can specify the default process for that container

→ The default process is modified in the command line itself.

eg:-

```
docker run --name U2 -P -it nginx bash
```

default process

→ Now bash will act as a default process of nginx container.

→ Now if you specify the default process in command line if it is working, it means that the designer of that dockerfile of that image has CMD or else ENTRYPOINT

→ ~~If it is ENTRYPOINT in command line it won't~~

eg:-

```
docker run --name U3 -P -it ubuntu date
```

now it will become default prog

→ As long as the default process is running container runs after that it goes into exited state.

Networks:-

→ Docker supports 4 types of networks.

① bridge n/w
② host n/w
③ null n/w
④ overlay n/w

① bridge n/w:-

→ when you create a multi container architecture on a single server, then all the containers by default linked with each other. i.e they can communicate with each other.

→ So I don't want to communicate one container on one network with another container. In this case customisation of n/w came into existence.

→ We can create our own networks.

② host network:-

→ Create one container and it must use the ip address of host machine, then you put that container on host n/w.

eg:- I want run a database container and no other containers put it on host n/w.

③ null network:-

→ The container which is created on this n/w will not communicate with other containers and host machine too. Then you put it on null n/w.

Scenario:-

→ They will create a container on bridge n/w. and they keep some confidential data in that container. after that they change bridge n/w to null, so that no one

can access it.

→ If they want to access the data again they change the null to bridge n/w, and access the data and immediately change it to null n/w.

→ This is done for docker security, as long as the container on null n/w no one can access it.

④ Overlay network:

→ when containers are running on multiple servers and they want to communicate with each other they depend on overlay n/w.

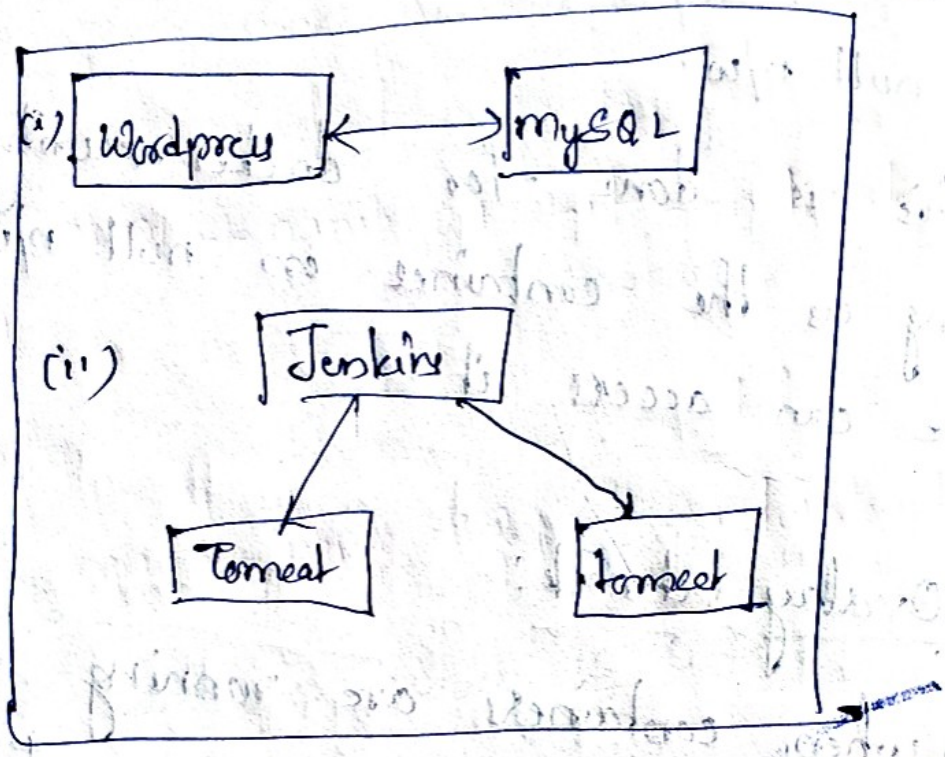
Bridge n/w:-

→ In this all the containers by default communicating with each other whether you use --link or not.

eg:- I want to create two architectures

(i) wordpress linked with mysql.

(ii) Jenkins linked with tomcat



→ In this p/w every container is talking with each other, the containers on the jenkins architecture are communicating with all the containers on wordpress architecture

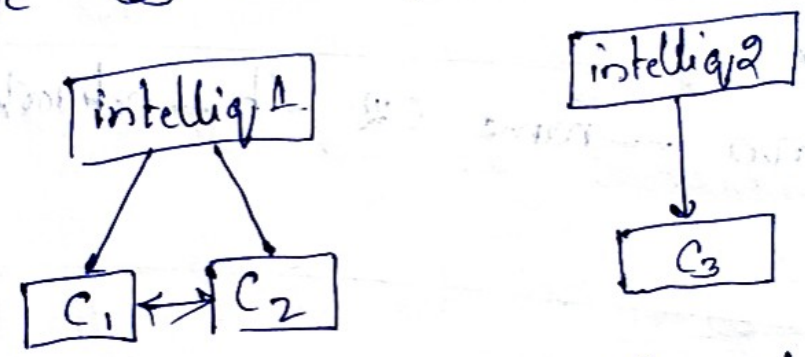
→ So I don't want that to be happened, therefore I go with unlink those architectures.

→ Now I can create customised n/w

Customised → bridge n/w:

Scenario:

① Create one network as intelliq1 and on this create two busybox containers and another n/w as intelliq2, on this create a busbox container;



→ See that containers on the intelliq1 should not talk with intelliq2.

→ Create a intelliq1 n/w

```
docker network create --driver bridge intelliq1
```

→ Create a intelliq2 n/w

```
docker network create intelliq2
```

→ In this if you don't give --driver, it

by default takes as bridge n/w

```
docker network ls
```

→ Now create a busy box container on intelliq1 n/w

```
docker run --name c1 -it --network intelliq1
busybox
```

→ ctrl P, Q

```
docker run --name c2 -it --network intelliq1
busybox
```

→ ctrl P, Q

→ Now create a busybox container on a intelliq2 n/w

```
docker run --name c3 -it --network intelliq2
busybox
```

→ ctrl P, Q

→ do ping c2 from c1, because they are on same n/w they can communicate

→ from c3 ping c1/c2, they can't communicate with each other because they are on different n/ws

(*) Now I can make c2 on intellij2, one container on one n/w can be present on any no. of n/ws.

→ Now c2 can communicate with c1 and c3, but c1 can't communicat with c3.

docker network connect intellij2 c2

(**) So one container can be present on any no. of n/w's

→ whenever we are creating n/w's, they are coming with one series of ip_address

→ eg:- In the previous scenario

172.18.0.2 c1
172.18.0.3 c2
172.19.0.2 c3

→ The sender is changing from one n/w to another n/w.

→ We can control that sender also

subnet :-

It is used to specify the ip address of the n/w.

→ We are using IPv4 format

→ In this ip addresses are created based on combinations we get 2^n

$n = 32 - a$

--subnet 10.0.0.0/a

$a \leq 32$.

eg:- docker network create --drive bridge --subnet 10.0.0.0/24 intelliq

$32 - 24 = 8$

$2^8 = 256$, ip-addresses we get.

10.0.0.0
1
2
:
255 → ip addresses

depending on the size of the infrastructure you create, for that size we create ip addresses.

→ Now create a container on intellij

```
docker run --name n1 -d --network intellij
nginx
```

for this the ip address starts we 10.0.0.1

Registry :-

→ We have two registries
  (i) public
  (ii) private

→ Public is docker hub, where everyone can download them.

→ Private :- only the certain people can download and upload : Container

eg:- ECR → Elastic Registry

How to push images into Public registry.

① Go to dockerhub and sign in.

② Now on your machine/docker host

```
docker login
```

Username :

password :

③ Now create a image as

```
docker build -t username/imagename
```

eg:- docker build -t sudarsbansw7/nginx455

④ Now push that image

```
docker push sudarsbansw7/nginx455
```

→ Private Registry :- I

→ You can create private registry in multiple ways.

(i) ECR → Elastic Container Registry → it is used in lots of companies as a private registry

Interview

→ Say that we are uploading the images into ECR

→ On docker hub also you can create private registry, on paid service.

→ For free you can create only one pvt registry, only one image is uploaded

How to create pvt repository

→ Click on create repository

→ Repository name as image_name

→ Select private option

→ Now create a image as shown in below

```
docker build -t username/repository name
```

eg!- docker build -t sudarshanswd/nginx445

→ Now push into repository

```
docker push sudarshanswd/nginx445
```

Private Registry - V

→ On docker hub private registry is restricted to only private image to be uploaded.

→ So docker has given one registry image. take that image and create container out of it, then that container behaves as a private registry.

→ Now create a registry with the image

```
docker run --name localregistry -d -p 5000:5000
registry
```

→ Now push any image into your private registry

```
docker tag alpine localhost:5000/alpine
```

→ Now,

```
docker push localhost:5000/alpine
```

** ECR - Private Registry :-

→ The ~~docker~~ dockerhost on which, the customised images or any images are present, to connect that host to ECR we have a service called IAM (Identity Access Management) it is nothing but user-administration i.e giving permissions to your instance on the ECR.

→ Search for IAM on AWS account

→ In that lefthand corner there is a option called Roles, select that

→ Click on create Role

→ Select AWS Service

→ Use case

→ Select EC2

→ Select first one

Allow EC2 instances to call AWS services on your behalf

→ Next

→ Select → Administrator Access → Next

→ Role details

Role name → give some name → create role

eg:- ECR_Private

→ Now this role has to be linked with dockerhost

→ Highlight the dockerhost instance → go to actions → security → modify IAM role

→ Select our Role → update IAM role

eg:- ECR_Private

→ Now search for ECR

→ Select ECR

→ Create repository → Private → Repository name i.e with what name the image has to be uploaded

eg:- myngnix

→ Now you created a repository.

To upload images into that repository

Select → View push commands → go through the command → copy and paste.

→ install aws on ubuntu instance

`apt install awscli`

then copy and paste these commands.

Image layers:-

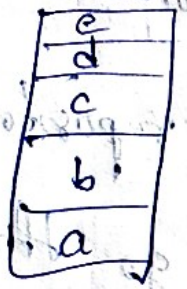
→ For every image there are layers, when you pull that image it downloads all those layers for first time

→ Next time you are downloading another image.

→ Then docker downloads only the unique layers of both images.

Lets say for tomcat we have 5 layers

tomcat



nginx — 8 layers

→ If some layers in tomcat and nginx are same then docker won't download those layers, it downloads other layers those are unique.

→ In windows to download skype, it downloads total setup, but docker downloads only unique layers for next software.

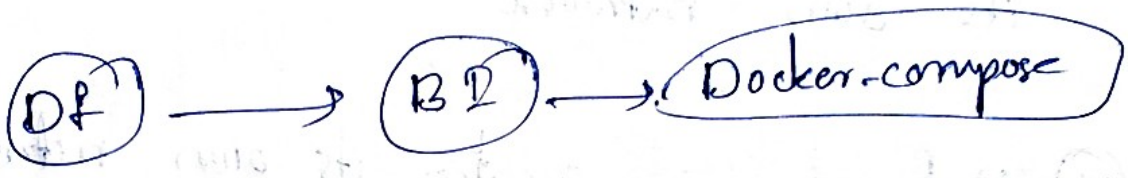
Docker compose :-

This is used for multi container architect -ure.

Using customised image in docker compose.

→ Create a image via. dockerfile

→ Use that image in docker-compose.yml.



→ Now for creating container directly we can use dockerfile in the docker compose.

→ Now dockerfile creates image and creates container out of it.

eg:- vim docker-compose.yml

```
version: '3.8'
services:
    myjenkins:
        build: .
        ports:
            - 5050:8080
```

→ The above compose file uses the dockerfile, it automatically builds the image and creates containers.

Networks :-

→ The docker-compose does not use default bridge network, but it creates its own network

(*) Docker compose creates its own network, but if created one network name as intelligit, then docker compose should create two containers on this network only ?

→ docker network create --driver bridge --subnet 10.0.0.0/24 intelligit

vim docker-compose.yml

```
---
version: '3.8'
services:
  mydb:
    image: mysql:5
    environment:
      MYSQL_ROOT_PASSWORD: sudarshan

  myword:
    image: wordpress
    ports:
      - 8888:80
networks:
  default:
    external:
      name: intelliqit
...
```

Syntax

```
networks:
  default:
    external:
      name: name of the n/w
```

(*) Create jenkins environment, where jenkins should run on abe n/w and qaserver and prodserver should run on xyz n/w.

```
version: '3.8'
services:
  myjenkins:
    image: jenkins/jenkins
    ports:
      - 5050:8080
    networks:
      - abe
  qaserver:
    image: tomcat
    ports:
      - 6060:8080
    networks:
      - xyz
  prodserver:
    image: tomcat
    ports:
      - 7070:8080
    networks:
      - xyz
networks:
  abe: { }
  xyz: { }
```

(*) Create separate volumes for mysql and wordpress.

```
---
version: '3.8'
services:
  mydb:
    image: mysql:5
    environment:
      MYSQL_ROOT_PASSWORD: sudarshan
    volumes:
      - db:/var/lib/mysql
                  path
  mywordpress:
    image: wordpress
    ports:
      - 8888:80
    volumes:
      - wordpress:/var/www/html
                  path
volumes:
  db:
  wordpress:
...
```

Note: get the volumes path for each container from google.

To deploy free html website with httpd

① `mkdir website`

→ Now go inside that folder

`cd website`

→ Here you put the html file or copy a free css website

`wget paste that url`

→ Now that file is ziped format

`unzip zippedfile_name`

→ Move into unzipped file, copy that unzipped all file into website folder

`cp -R * ../.`

→ Now create a docker file

vim dockerfile

FROM httpd

COPY website /usr/local/apache2/htdocs/

→ Create a container from that image