DOCKER

Virtualization

- This is the process of running multiple OS's parallelly on a single piece of h/w. Here we have h/w(bare metal) on top of which we have host Os and on the host Os we install an application called as hypervisor on the hypervisor we can run any no of OSs as guest OS

- The disadvantage of this approach is this application running on the guest OS have to pass through n number of lavers to access the H/W resources.

Containerization

- Here we have bare metal on top of which we install the host Os and on the host OS we install an application called as Docker Engine On the docker engine we can run any application 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

- These containers pass through less no of layers to access the h/w resources also organizations need not spend money on purchasing licenses of different OS's to maintain various applications Docker can be used at the the stages of S/W development life cycle
- Build---->Ship--->Run

- **Docker comes in 2 flavours**
    - Docker CE (Community Edition)
    - Docker EE (Enterprise Edition)

Setup of Docker on Windows

1 Download docker desktop from
 https://www.docker.com/products/docker-desktop
2 Install it
3 Once docker is installed we can use Power shell
 to run the docker commands

Install docker on Linux

1 Create an Ubuntu instance on AWS
2 Connect to it using git bash
3 Execute the below 2 commands
 curl -fsSL https://get.docker.com -o get-docker.sh
 sh get-docker.sh

Images and Containers

➜ 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 Client:** This is the CLI of docker where the user can execute the docker commands, The docker client accepts these commands and passes them to a background process called "docker daemon"

- **Docker deamon:** This process accepts the commands coming from the docker client and routes them to work on docker images or containers or the docker registry

- **Docker registry**: This is the cloud site of docker where docker images are stored. This is of two types
  ➢ 1 Public Registry(hub.docker.com)
  ➢ 2 Private Registry(Setup on one of our local servers)

Important docker commands

Working on docker images

1 To pull a docker image
  docker pull image_name

2 To search for a docker images
  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
  or
  docker image ls
5 To get detailed info about a docker image
  docker image inspect image_name/image_id

6 To delete a docker image that is not linked to any container
  docker rmi image_name/image_id

7 To delete an 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 image
  docker system prune -af

Madhav

11 To create a docker image from a dockerfile
   docker build -t image_name .

12 To create an image from a customised container
   docker commit container_id/container_name image_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_id/container_name

16 To stop a container
   docker stop container_id/container_name

17 To restart a container
   docker restart container_id/container_name
   To restart after 10 seconds
   docker restart -t 10 container_id/container_name

18 To delete a stopped container
   docker rm container_id/container_name

19 To delete a running container
   docker rm -f container_id/container_name

20 To stop all running container
   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_id/container_name
24 To see the logs generated by a container
   docker logs container_id/container_name

25 To create a docker container
   docker run image_name/image_id

DOCKER

Run command options

➔ **--name:** Used to give a name to the container

➔ **--restart:** Used to keep the container in running 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 mapped with some host port that is greater than 30000

➔ **--link :** Used to create a link between multiple containers to create a   microservices architecture.

➔ **--network:** Used to start a container on a specific 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 cpus 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_id/container_name

27 To run any process in a container from outside the container
   Docker exec -it container_id/container_name process_name

   Eg: To run the bash process in a container
      docker exec -it container_id/container_name 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_id/container_name

30 To see the processes running in a container
   docker container container_id/container_name 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/container_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

Day 3
Use Case 1

Create an nginx container in detached mode and name its webserver
Also perform port mapping

docker run --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_of_dockerhost:8888

Use Case 2

➔ Start tomcat as a container and perform automatic port mapping
      docker run --name appserver -d -P tomee

➔ To see the ports used by the above container
      docker port appserver

➔ To access the httpd from browser
    piblic_ip_of_dockerhost:9090

Use Case 3

➔ Start a 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 container
      docker port appserver

➔ To access jenkins from browser
      public_ip_of_docker_host:port_no_from_above_command

Use Case

➔ Create a mysql container and login as root user and create some SQL tables

1 Create a mysql container
  docker run --name db -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql:5

2 To check if the mysql container is running
  docker container ls

3 To go into the bash shell of the container
  docker exec -it db bash

4 To login into the database
  mysql -u root -p
  Password: intelliqit

5 To see the list of databases
  show databases;

6 To move into any of the above database
  use database name;
  Eg: use sys;

7 To create emp and dept tables here Open
  https://justinsomnia.org/2009/04/the-emp-and-dept-tables-for-mysql/

 Copy script from emp and dept tables creation
  Paste in the mysql container

8 To see the data of the tables
  select * from emp;
  select * from dept;

Use Case

➔ Create an ubuntu container and launch interactive terminal
    docker run --name u1 -it ubuntu

➔ To come out of the **cento's** container
    exit

Use Case

➔ Start centos as a container and launch interactive terminal in it
    docker run --name mycentos -it centos

➔ To come out of the **cento's** container
    exit

To setup a multi container architecture

1 Using --link run command option (deprecated)
2 Docker compose
3 Docker **Networking's**

4 Python Scripting
5 Ansible Playbooks

Use Case

➔ **Create 2 busybox containers c1 and c2 and link them**

1 Create a busybox container and name it c1
  docker run --name c1 -it busybox

2 To come out of the c1 container without exit
  ctrl+p,ctrl+q

3 Create another busybox container c2 and link it with c1 container
  docker run --name c2 -it --link c1:mybusybox busybox

4 Check if c2 is pinging to c1
  ping c1

Use Case

➔ **Setup wordpress and link it with mysql container**

1 Create a mysql container
docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql:5

2 Create a wordpress container and link with the mysql container
docker run --name mywordpress -d -p 8888:80 --link mydb:mysql wordpress

3 To check if wordpress and mysql containers are running
docker container ls

4 To access wordpress from a browser
public_ip_dockerhost:8080

5 To check if wordpress is linked with mysql
docker inspect mywordpress
Search for "Links" section

Use Case

➔ **Setup CI-CD environment where a Jenkins container is linked with 2 tomcat containers for Qaserver and Prodserver**

1 Create a jenkins container
docker run --name myjenkins -d -p 5050:8080 jenkins/jenkins

2 To access jenkins from browser
public_ip_dockerhost:5050

3 Create a tomcat container as qaserver and link with jenkins container
docker run --name qaserver -d -p 6060:8080 --link myjenkins: jenkins tomee

4 Create another tomcat container as prodserver and link with jenkins
docker run --name prodserver -d -p 7070:8080 --link myjenkins:jenkins tomee

5 Check if all 3 containers are running
docker container ls

Setup LAMP architecture

1 Create mysql container
docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql

2 Create an Apache container and link with mysql container
docker run --name apache -d -p 9999:80 --link mydb:mysql httpd

3 Create a php container and link with mysql and apache containers
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

Create Postgres with adminer client

1 Create a Postgres application
  docker run --name mydb -d -e POSTGRES_PASSWORD=intelliqit -e POSTGRES_DB=mydb -e POSTGRES_USER=myuser Postgres

2 Create an adminer application
  docker run --name myadminer -d -p 9090:80 --link mydb:postgres adminer

3 To access the db from adminer Launch browser
  public_ip_of_dockerhost:9090

UseCase

➜ **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 deprecated and the same individual command have to be given multiple times to setup similar architectures. 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

UseCase

➔ **Create a docker compose file to setup a mysql and wordpress container and link them**

vim docker-compose. Yml

```
---
version: '3.8'

services:
 mydb:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit

 mywordpress:
  image: wordpress
  ports:
   - 8888:80
  links:
   - mydb:mysql
...
```

➔ To setup the containers from the above file
   docker-compose up -d

➔ To stop all the container of the docker compose file
   docker-compose stop

➔ To start the container
   docker-compose start

➔ To stop and delete
   docker-compose down

UseCase

➔ **Create a docker compose file to setup the CI-CD environment where a jenkins container is linked with 2 tomee containers one for qaserver and other for prodserver**

vim docker-compose.yml

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

 qaserver:
  image: tomee
  ports:
   - 6060:8080
  links:
   - myjenkins:jenkins

 prodserver:
  image: tomee
  ports:
   - 7070:8080
  links:
   - myjenkins:jenkins
```

UseCase

➔ **Create a docker compose file to setup the LAMP architecture**

vim lamp.yml
```
---
version: '3.8'
services:
 mydb:
  image: mysql
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit

 apache:
  image: httpd
  ports:
   - 8989:80
  Links
   - mydb:mysql

 php:
```

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

➔ To create containers from the above file
   docker-compose -f lamp.yml up -d

➔ To delete the containers
   docker-compose -f lamp.yml down

UseCase

➔ **Create a docker compose file to setup the selenium testing environment where a selenium hub container is linked with 2 node containers one with chrome and other with firefox**

vim docker-compose.yml

```
---
version: '3.8'
services:
 hub:
  image: selenium/hub
  ports:
   - 4444:4444
  container_name: hub
 chrome:
  image: selenium/node-chrome-debug
  ports:
   - 5901:5900
  links:
   - hub:selenium
  container_name: chrome
 firefox:
  image: selenium/node-firefox-debug
  ports:
   - 5902:5900
  links:
   - hub:selenium
  container_name: firefox
...
```

➔ To setup the above architecture
   docker-compose up -d

➔ To check the running containers
   docker container ls

➔ To delete the containers
   docker-compose down

Docker Volumes

➔ Containers are ephemeral(temporary) but the data processed by the containers should be persistent. Once a container is deleting all the data of the container is lost to preserve the data even if the container is deleted, we can use volumes

Volumes are classified into 3 types

1 Simple docker volume
2 Sharable docker volumes
3 Docker volume containers

Simple Docker volumes

➔ These volumes are used only for preserving the data on the host machine even if the containers are deleted

UseCase
➔ Create a directory /data and mount it as a volume on an ubuntu container
➔ Create some files in the mounted volumes and check if the files are preserved on the host machine even after the container is deleted

1 Create /data directory
  ➢ mkdir /data

2 Create an ubuntu container and mount the above directory as volume
  ➢ docker run --name u1 -it -v /data ubuntu

➔ In the container u1 go into /data directory and create some files
  ➢ cd /data
  ➢ touch file1 file2 file3
  ➢ exit

3 Identify the location where the mounted data is preserved
  ➢ docker inspect u1
  ➢ Search for "Mounts" section and copy the "Source" path

4 Delete the container
  ➢ docker rm -f u1

5 Check if the data is still present
  ➢ cd "source_path_from_step3"
  ➢ ls

Sharable Docker volumes

➔ These volumes are sharable between multiple containers
➔ Create 3 centos containers c1,c2,c3.
➔ Mount /data as a volume on c1 container ,c2 should use the volume used by c1 and c3 should use the volume used by c2

1 Create a centos container c1 and mount /data
　docker run --name c2 -it -v /data centos

2 Go into the data folder create files in data folder
　cd data
　touch f1 f2

3　Come out of the container without exit
　ctrl+p,ctrl+q

4 Create another centos container c2 and it should use the volumes used by c1
　docker run --name c2 -it --volumes-from c1 centos

5 In the c2 container go into data folder and create some file
　cd data
　touch f3 f4

6 Come out of the container without exit
　ctrl+p,ctrl+q

7 Create another centos container c3 and it should use the volume used by c2
　docker run --name c3 -it --volumes-from c2 centos

8 In the c3 container go into data folder and create some file
　cd data
　touch f5 f6

9 Come out of the container without exit
　ctrl+p,ctrl+q

10 Go into any of the 3 containers and we will see all the files
　docker attach c1
　cd /data --➔ ls
　　exit

12 Identify the location the where the mounted data is stored
　docker inspect c1
　Search for "Mounts" section and copy the "Source" path

13 Delete all containers
　docker rm -f c1 c2 c3

14 Check if the files are still present
　cd "source_path_from"step12"

Docker volume containers

➔ **These volumes are bidirectional i.e. the changes done on host will be reflected into container and changes done by container will be reflected to host machine**

1 Create a volume
  docker volume create myvolume

2 To check the location where the mounted the volume works
  docker volume inspect myvolume

3 Copy the path shown in "MountPoint" and cd to that Path
  cd "MountPoint"

4 Create few files here
  touch file1 file2

5 Create a centos container and mount the above volume into the tmp folder
  docker run --name c1 -it -v myvolume:/tmp centos

6 Change to tmp folder and check for the files
  cd /tmp
  ls

➔ **If we create any files here, they will be reflected to host machine and these files will be present on the host even after deleting the   container**.

UseCase

➔ **Create a volume "newvolume" and create tomcat-users.xml file in it**
➔ **Create a tomcat container and mount the above volume into it Copy the tomcat-users.xml files to the required location**

1 Create a volume
  docker volume create newvolume

2 Identify the mount location
  docker volume inspect newvolume
  Copy the "MountPoint" path

3 Move to this path
  cd "MountPoint path"

4 Create a file called tomcat-users.xml
  cat > tomcat-users.xml
  <tomcat-users>
    <user username="intelliqit" password="intelliqit" roles="manager-script"/>
  </tomcat-users>

5 Create a tomcat container and mount the above volume
  docker run --name webserver -d -P -v newvolume:/tmp tomcat

6 Go into bash shell of the tomcat container
  docker exec -it webserver bash

7 Move the tomcat-users.xml file into conf folder
  mv /tmp/tomcat-users.xml conf/

Creating customised docker images

➔ This can be done in 2 ways
   1) Using docker commit command
   2) Using dockerfile

Using the docker commit command

UseCase

➔ **Create an ubuntu container and install some s/w's in it Save this container as an image and later create a new container from the newly created image. We will find all the s/w's that we installed.**

1 Create an ubuntu container
  docker run --name u1 -it ubuntu

2 In the container update the apt repo and install s/w's
  apt-get update
  apt-get install -y git

3 Check if git is installed or not
  git --version
  exit

4 Save the customised container as an image
  docker commit u1 myubuntu

5 Check if the new image is created or not
  docker images

6 Delete the previously create ubuntu container
  docker rm -f u1

7 Create an new container from the above created image
  docker run --name u1 -it myubuntu

8 Check for git
  git --version

Dockerfile

➔ **Dockerfile uses predefined keyword to create customised docker images.**

Important keyword in dockerfile

FROM : This is used to specify the base image from where a customised docker image has to be created

MAINTAINER : This represents the name of the organization or the author that has created this dockerfile

RUN : Used to run Linux commands in the container Generally it used to do s/w installation or
running scripts

USER : This is used to specify who should be the default user to login into the container

COPY : Used to copy files from host to the customised image that we are creating

ADD : This is similar to copy where it can copy files from host to image but ADD can also download files from .
some remote server

EXPOSE : Used to specify what port should be used by the container

VOLUME : Used for automatic volume mounting i.e. we will have a volume mounted automatically when the　　　.
container start

WORKDIR : Used to specify the default working directory of the container

ENV : This is used to specify what environment variables should be used

CMD : Used to run the default process of the container from outside

ENTRYPOINT : This is also used to run the default process of the container

LABEL: Used to store data about the docker image in key value pairs

SHELL : Used to specify what shell should be by default used by the image

UseCase

➔ Create a dockerfile to use nginx as base image and specify the maintainer as intelliqit

1 Create docker file
vim dockerfile

FROM nginx
MAINTAINER intelliqit

2 To create an image from this file
 docker build -t mynginx .

3 Check if the image is created or not
  docker images

UseCase

➔ Create a dockerfile from ubuntu base image and install   git in it
   1 Create dockerfile
    vim dockerfile
    FROM ubuntu
    MAINTAINER intelliqit
    RUN apt-get update
    RUN apt-get install -y git

   2 Create an image from the above file
    docker build -t myubuntu .

   3 Check if the new image is created
    docker images

   4 Create a container from the new image and it should have git installed
    docker run --name u1 -it myubuntu
    git --version

Cache Busting

➔ When we create an image from a dockerfile docker stores all the executed instructions in a its cache. Next time if we edit the same docker file and add few new instructions and build an image out of it docker will not execute the previously executed statements Instead it will read them from the cache This is a time saving mechanism The disadvantage is if the docker file is edited with a huge time gap then we might end up installing s/w's that are outdated

Eg:
FROM ubuntu
RUN apt-get update
RUN apt-get install -y git

➔ If we build an image from the above dockerfile docker saves all these instructions in the docker cache and if we add the below statement
   RUN apt-get install -y tree
   only this latest statement will be executed

➔ To avoid this problem and make docker execute all the instructions once more time without reading from cache we use "cache busting" docker build --no-cache -t myubuntu .

DOCKER

Madhav

➔ **Download docker shell script into the docker host and copy it into the customised docker image and later install it at the time of creating the image**

1 Download docker script
  curl -fsSL https://get.docker.com -o get-docker.sh

2 Create the dockerfile
  vim dockerfile
  FROM ubuntu
  MAINTIANER intelliqit
  COPY get-docker.sh /
  RUN sh get-docker.sh

4 Create an image from the dockerfile
  docker build -t myubuntu .

5 Create a container from the above image
  docker run --name u1 -it myubuntu

6 Check if the get-docker.sh is present in / and also see if docker is installed
  docker --version

➔ **Create a dockerfile to create an ansible image**
1 vim dockerfile
  FROM ubuntu
  RUN apt-get update
  RUN apt-get install -y software-properties-common
  RUN apt-get install -y ansible

2 Build an image
  docker build -t ansible
  This container will have ansible installed in it

➔ **Create a dockerfile from ubuntu base image and download jenkins.war into it**

1 Create a dockerfile
  vim dockerfile
  FROM ubuntu
  MAINTIANER intelliqit
  ADD https://get.jenkins.io/war-stable/2.263.4/jenkins.war /

2 Create an image from the above dockerfile
  docker build -t myubuntu .

4 Create a container from this image
  docker run --name u1 -it myubuntu

5 Check if jenkins.war is present
  ls

DOCKER

➔ **Create a dockerfile from jenkins base image and make the default user as root**

1 vim dockerfile
FROM jenkins/jenkins
MAINTAINER intelliqit
USER root

2 Create an image from the above dockerfile
docker build -t myjenkins .

3 Create a container from the above image
docker run --name j1 -d -P myjenkins

4 Go into the interactive shell and check if the default user is root
docker exec -it j1 bash
whoami

➔ **Create a dockerfile from nginx base image and expose 90 ports**

1 vim dockerfile
FROM nginx
MAINTAIENR intelliqit
EXPOSE 90

2 Create an image from the above file
docker build -t mynginx .

3 Create a container from the above image
docker run --name n1 -d -P mynginx

4 To check the port
docker port n1

➔ **Create docker image with a volume on an image**

1 Create a dockerfile
vim dockerfile
FROM ubuntu
MAINTAINER intelliqit
VOLUME /data

2 Create an image
docker build -t myubuntu .

3 Create a container
docker run --name u1 -it myubuntu
cd data
touch file1 file2
exit

4 Delete container
docker rm -f u1

Files will be available

UseCase

1) **Create a dockerfile from ubuntu base image and make if behave like jenkins**

vim dockerfile
FROM ubuntu
MAINTAINER intelliqit
RUN apt-get update
RUN apt-get install -y openjdk-11-jdk
ADD https://get.jenkins.io/war-stable/2.426.2/jenkins.war /
ENTRYPOINT ["java","-jar","jenkins.war"]

2 Create an image
   docker build -t myubuntu .

3 Create a container
   docker run --name u1 -it myubuntu
➔   the container behaves like jenkins

UseCase

➔ **Create a dockerfile from ubuntu base image and make it behave like nginx**

1 Create a dockerfile

 vim dockerfile

 FROM ubuntu
 MAINTAINER intelliqit
 RUN apt-get update
 RUN apt-get install -y nginx
 ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
 EXPOSE 80

2 Create an image from the above dockerfile
   docker build -t myubuntu .

3 Create a container from the above image and it will work like nginx
   docker run --name n1 -d -P myubuntu

4 Check the ports used by nginx
   docker container ls

5 To access nignx from browser
   public_ip_of_dockerhost:port_no_captured_from_step4

CMD and ENTRYPOIT

➔ Bothe of them is used to specify the default process that should be triggered when the container starts but the CMD instruction can be overridden with some other process passed at the docker run command

Eg:
FROM ubuntu
RUN apt-get update
RUN apt-get install -y nginx
CMD ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80

➔ Though the default process is to trigger nginx we can bypass that and make it work on some other process

docker build -t myubuntu .
➔ Create a container
   docker run --name u1 -it -d myubuntu

Here if we inspect the default process, we will see that nginx as the default process

docker container ls

➔ on the other hand, we can modify that default process to something else
   docker run --name u1 -d -P myubuntu ls -la
➔ Now if we do "docker container ls" we will see the default process to be "ls -la"

Docker Networking

🔶 **Docker supports 4 types of networks**

1 Bridge
2 Host
3 Null
4 Overlay

UseCase

🔶 Create 2 bridge networks intelliq1 and intelliq2
🔶 Create 2 busybox containers c1,c2 and c3
🔶 c1 and c2 should run on intelliq1 network and should ping each other
🔶 c3 should run on intelliq2 network and it should not be able to ping c1 or c2
🔶 Now put c2 on intelliq2 network, since c2 is on both intelliq1 and intelliq2
🔶 networks it should be able to ping to both c1 and c3 but c1 and c3 should not ping each other directly

1 Create 2 bridge networks
   docker network create --driver bridge intelliq1
   docker network create --driver bridge intelliq2

2 Check the list of available networks
   docker network ls

3 Create a busybox container c1 on intelliqi1 network
   docker run --name c1 -it --network intelliq1 busybox
   Come out of the c1 container without exit ctrl+p,ctrl+q

4 Identify the Ip address of c1
   docker inspect c1

5 Create another busybox container c2 on intelliq1 network
   docker run --name c2 -it --network intelliq1 busybox
   ping ipaddress_of_c1   (It will ping)
   Come out of the c2 container without exit ctrl+p,ctrl+q

6 Identify the Ip address of c2
   docker inspect c2

7 Create another busybox container c3 on intelliq2 network
   docker run --name c3 -it --network intelliq2 busybox
   ping ipaddress_of_c1 (It should not ping)
   ping ipaddress_of_c2 (It should not ping)
   Come out of the c3 container without exit ctrl+p,ctrl+q

8 Identify the Ip address of c3
   docker inspect c3

9 Now attach intelliq2 network to c2 container
   docker network connect intelliq2 c2

10 Since c2 is now on both intelliq1 and intelliq2 networks it should ping
    to both c1 and c3 containers
    docker attach c2
    ping ipaddress_of_c1 (It should ping)
    ping ipaddress_of_c3 (It should ping)
    Come out of the c2 container without exit ctrl+p,ctrl+q

11 But c1 and c3 should not ping each other
    docker attach c3
    ping ipaddress_of_c1 (It should not ping)

Working on docker registry

DOCKER

⚜ **This is the location where the docker images are saved This is of 2 types**
➔ 1 Public registry
➔ 2 Private registry

UseCase

➔ **Create a customised centos image and upload into the public registry**

1 Signup into hub.docker.com

2 Create a customised docker image from dockerfile
 docker build -t intelliqit/nginx19 .

3 Push to registry
 docker push intelliqit/nginx19

===
Day 12
===

Private Registry

ECR

- 1 Create an IAM role with admin privileges and assign to docker host

- 2 Search for ECR service on aws and create a private ecr registry

- 3 Click on View push command and copy paste the command in the docker host

Note: To create network with a specific subnet range
> **network create --driver bridge --subnet=192.168.2.0/24 intelliqit**

➔ **Docker compose by default creates its own customised bridge network and creates containers on the network**

vim docker-compose.yml

```
---
version: '3.8'
services:
 mydb:
  image: postgres
  environment:
    POSTGRES_PASSWORD: intelliqit
    POSTGRES_DB: mydb
    POSTGRES_USER: myuser

 adminer:
  image: adminer
  ports:
   - 8080:8080
```

➔ To setup the containers
> **docker compose up -d**

➔ To see the list of containers
> **docker container ls**

➔ To see the list of networks
> **docker network ls**

➔ To above 2 containers will be running on a new bridge network that is created by docker compose

➔ To delete the containers
> **docker compose down**

➔ This will not only delete the containers it will also delete the networks that got created.

Madhav

UseCase

➔ **Create a custom bridge network and create a docker compose file to start postgres and adminer container on the above created network**

1 Create a custom bridge network
  docker network create --driver bridge --subnet 10.0.0.0/24 intelliqit

2 Create a docker compose file
  vim docker-compose.yml
---
version: '3.8'
services:
 db:
  image: postgres
  environment:
   POSTGRES_PASSWORD: intelliqit
   POSTGRES_USER: myuser
   POSTGRES_DB: mydb

 adminer:
  image: adminer
  ports:
   - 8888:8080

networks:
 default:
  external:
   name: intelliqit
...

3 To create the containers
  docker-compose up -d

4 To see if adminer and postgres containers are created
  docker container ls

5 To check if they are running on intelliqit network
  docker inspect container_id_from_Step4

➔ **Create a dockerfile and use it directly in docker-compsoe**

vim dockerfile

FROM jenkins/jenkins
MAINTAINER intelliqit
RUN apt-get update
RUN apt-get install -y git

vim docker-compose.yml
version: '3.8'

services:
 jenkins:
  build: .
  ports:
   - 7070:8080

 mytomcat:
  image: tomee
  ports:
   - 6060:8080

...

To start the services
docker-compose up

➔ **Docker compose file to create 2 networks and run containers on different network**

vim docker-compose.yml
---
version: '3.8'
services:
 mydb:
  image: jenkins/jenkins
  ports:
   - 5050:8080
  networks:
   - abc

 qaserver:
  image: tomee
  ports:
   - 6060:8080
  networks:
   - xyz

 prodserver:
  image: tomee
  ports:

```
   - 7070:8080
  networks:
   - xyz

networks:
 abc: {}
 xyz: {}
...
```

➔ **Docker compose file to create 2 containers and also create 2 volumes for both the containers**

```
---
version: '3.8'
services:
 db:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit
  volumes:
   mydb:/var/lib/mysql

 wordpress:
  image: wordpress
  ports:
   - 9999:80
  volumes:
   wordpress:/var/www/html

volumes:
 mydb:
 wordpress
```

To start the service
docker-compose up -d

To see the list of volumes
docker volume ls

===

Container orchestration

➔ **Container orchestration** is the process of managing multiple containers in a way that ensures they all run at their best. This can be done through **container orchestration tools**, which are software programs that automatically manage and monitor a set of containers on a single machine or across multiple machines[1]. Here are some key points about container orchestration:

1. **Definition**: Container orchestration automates various tasks related to containers, including provisioning, deployment, networking, scaling, availability, and lifecycle management.

2. **Importance**: As organizations adopt containerization, the number of containerized applications grows rapidly. Managing them manually becomes impossible without automation, especially within continuous integration/continuous delivery (CI/CD) or DevOps pipelines.

3. **Popular Platform**: **Kubernetes** is currently the most widely used container orchestration platform. Major public cloud providers, such as Amazon Web Services (AWS), Google Cloud Platform, IBM Cloud, and Microsoft Azure, offer managed Kubernetes services.

4. **Other Tools**: Besides Kubernetes, there are other container orchestration tools like **Docker Swarm** and **Apache Mesos**.

5. **How It Works**:

   o **Configuration Model**: Most container orchestration tools follow a declarative configuration model. Developers write configuration files (in YAML or JSON) that define the desired state of the application.

   o The configuration file typically includes information about container images, their locations (registries), resource provisioning, network connections, and security settings.

   o **Deployment Scheduling**: The orchestration tool schedules container deployment (and replicas for resiliency) to suitable hosts based on available CPU capacity, memory, and other specified requirements.

6. **Automation Benefits**: Container orchestration streamlines tasks such as scaling, maintenance, deployment, and network configuration. It ensures that containers run efficiently and communicate effectively with each other[2345].

➔ In summary, container orchestration simplifies the management of container lifecycles, making it easier to handle complex, dynamic environments where containerized software and applications thrive.

Docker Swarm

Docker Swarm is a container orchestration tool that simplifies the deployment and management of containerized applications in a clustered environment. Here are some key concepts and features of Docker Swarm:

1. **Cluster Management Integrated with Docker Engine**:
   - Docker Swarm allows you to create a swarm of Docker Engines (nodes) without needing additional orchestration software.
   - You can deploy application services to the swarm using the Docker CLI.
   - The decentralized design means that specialization between node roles (managers and workers) is handled at runtime, allowing you to build an entire swarm from a single disk image.
2. **Declarative Service Model**:
   - Define the desired state of services in your application stack using a declarative approach.
   - For example, you can describe an application with a web front end service, message queueing services, and a database backend.
3. **Scaling**:
   - Specify the number of tasks (replicas) you want to run for each service.
   - The swarm manager automatically adapts by adding or removing tasks to maintain the desired state.
4. **Desired State Reconciliation**:
   - The swarm manager constantly monitors the cluster state and reconciles differences between the actual state and the desired state.
   - If a worker machine hosting replicas crashes, the manager creates new replicas to replace them.
5. **Multi-Host Networking**:
   - Specify an overlay network for your services.
   - The swarm manager assigns addresses to containers on the overlay network during initialization or updates.
6. **Service Discovery**:
   - Each service in the swarm is assigned a unique DNS name by the swarm manager.
   - Load balancing is handled, and you can query every container running in the swarm through an embedded DNS server.
7. **Load Balancing**:
   - Expose service ports to an external load balancer.
   - Internally, specify how to distribute service containers between nodes.
8. **Secure by Default**:
   - TLS mutual authentication and encryption secure communications between swarm nodes.

Docker Swarm is a powerful tool for managing multiple Docker containers across different machines, providing a robust and efficient way to orchestrate containerized applications

Setup of Docker Swarm

1 Create 3 AWS ubuntu instances
2 Name them as Manager,Worker1,Worker2
3 Install docker on all of them
4 Change the hostname
  vim /etc/hostname
  Delete the content and replace it with Manager or Worker1 or Worker2
5 Restart
  init 6
6 To initialise the docker swarm
  Connect to Manager AWS instance
  docker swarm init
  This command will create a docker swarm and it will also generate
  a tokened
7 Copy and paste the token id in Worker1 and Worker2

==
➔ TCP port 2376 for secure Docker client communication. This port is required for Docker Machine to work. Docker Machine is used to orchestrate Docker hosts.

➔ TCP port 2377. This port is used for communication between the nodes of a Docker Swarm or cluster. It only needs to be opened on manager nodes.

➔ TCP and UDP port 7946 for communication among nodes (container network discovery).
➔ UDP port 4789 for overlay network traffic (container ingress networking).
==

Load Balancing:

➔ Each docker containers has a capability to sustain a specific user load. To increase this capability, we can increase the number of replicas(containers) on which a service can run

UseCase
➔ Create nginx with 5 replicas and check where these replicas are running

1 Create nginx with 5 replicas
  docker service create --name webserver -p 8888:80 --replicas 5 nginx

2 To check the services running in swarm
  docker service ls

3 To check where these replicas are running
  docker service ps webserver

4 To access the nginx from browser
  public_ip_of_manager/worker1/worker2:8888

5 To delete the service with all replicas
  docker service rm webserver

UseCase
➔ **Create mysql with 3 replicas and also pass the necessary environment variables**

1 docker service create --name db --replicas 3
        -e MYSQL_ROOT_PASSWORD=intelliqit mysql:5

2 To check if 3 replicas of mysql are running
  docker service ps db

Day 14

Scaling

➔ **This is the process of increasing the number of replicas or decreasing the replicas count based on requirement without the end user experiencing any down time.**

UseCase
➔ **Create httpd with 4 replicas and scale it to 8 and scale it down to 2**

1 Create httpd with 4 replicas
  docker service create --name websserver -p 9090:8080 --replicas 4 httpd

2 Check if 4 replicas are running
  docker service ps webserver

3 Increase the replicas count to 8
  docker service scale webserver=8

4 Check if 8 replicas are running
  docker service ps webserver

5 decrease the replicas count to 2
  docker service scale webserver=2

6 Check if 2 replicas are running
  docker service ps webserver

Rolling updates

➔ **Services running in docker swarm should be updated from once version to other without the end user downtime**

UseCase

➔ Create redis:3 with 5 replicas and later update it to redis:4 also, rollback to redis:3

1 Create redis:3 with 5 replicas
  docker service create --name myredis --replicas 5 redis:3

2 Check if all 5 replicas of redis:3 are running
  docker service ps myredis

3 Perform a rolling update from redis:3 to redis:4
  docker service update --image redis:4 myredis

4 Check redis:3 replicas are shut down and in tis place redis:4 replicas are running
  docker service ps myredis

5 Roll back from redis:4 to redis:3
  docker service update --rollback myredis

6 Check if redis:4 replicas are shut down and, in its place, redis:3 is running
  docker service ps myredis

===

➔ To remove a worker from swarm cluster
  docker node update --availability drain Worker1

➔ To make this worker rejoin the swarm
  docker node update --availability active Worker1

➔ To make worker2 leave the swarm
  Connect to worker2 using git bash
  docker swarm leave

➔ To make manager leave the swarm
  docker swarm leave --force

➔ To generate the token id for a machine to join swarm as worker
  docker swarm join-token worker

➔ To generate the token id for a machine to join swarm as manager
  docker swarm join-token manager

➔ To promote Worker1 as a manager
  docker node promote Worker1

➔ To demote "Worker1" back to a worker status
  docker node demote Worker1

FailOver Scenarios of Workers

➔ Create httpd with 6 replicas and delete one replica running on the manager Check if all 6 replicas are still running

➔ Drain Worker1 from the docker swarm and check if all 6 replicas are running on Manager and Worker2,make Worker1 rejoin the swarm

➔ Make Worker2 leave the swarm and check if all the 6 replicas are running on Manager and Worker1

1 Create httpd with 6 replicas
  docker service create --name webserver -p 8888:80 --replicas 6 httpd

2 Check the replicas running on Manager
  docker service ps webserver | grep Manager

3 Check the container id
  docker container ls

4 Delete a replica
  docker rm -f container_id_from_step3

5 Check if all 6 replicas are running
  docker service ps webserver

6 Drain Worker1 from the swarm
  docker node update --availability drain Worker1

7 Check if all 6 replicas are still running on Manager and Worker2
  docker service ps webserver

8 Make Worker1 rejoin the swarm
  docker node update --availability active Worker1

9 Make Worker2 leave the swarm
  Connect to Worker2 using git bash
  docker swarm leave
  Connect to Manager

10 Check if all 6 replicas are still running
   docker service ps webserver

FailOver Scenarios of Managers

➔ If a worker instance crashes all the replicas running on that worker will be moved to the Manager or the other workers.

➔ If the Manager itself crashes the swarm becomes headless i.e., we cannot perform container orchestration activities in this swarm cluster

➔ To avoid this, we should maintain multiple managers Manager nodes have the status as Leader or Reachable

➔ If one manager node goes down other manager becomes the Leader Quorum is responsible for doing this activity and if uses a RAFT algorithm for handling the failovers of managers. Quorum also is responsible for maintaining the min number of manager

➔ Min count of manager required for docker swarm should be always more than half of the total count of Managers

Total Manager Count	-	Min Manager Required	-	Fault Tolerance
1	-	1	-	0
2	-	2	-	0
3	-	2	-	1
4	-	3	-	1
5	-	3	-	2
6	-	4	-	2
7	-	4	-	3
8	-	5	-	3

==

Madhav

Overlay network

➔ **This is the default network used by docker swarm when container run multiple servers and the name of this network is ingress.**

UseCase

➔ Create 2 overlay networks intelliqit1 and intelliqit2
➔ Create httpd with 5 replicas on intelliqit1 network
➔ Create tomcat with 5 replicas on default overlay "ingress" network and later perform rolling network update to intelliqit2 network

1 Create 2 overlay networks
  docker network create --driver overlay intelliqit1
  docker network create --driver overlay intelliqit2

2 Check if 2 overlay networks are created
  docker network ls

3 Create httpd with 5 replicas on inteliiqit1 network
  docker service create --name webserver -p 8888:80 --replicas 5
                     --network intelliqit1 httpd

4 To check if httpd is running on intelliqit1 network
  docker service inspect webserver
➔ This command will generate the output in JSON format

➔ To see the above output in normal text format
        docker service inspect webserver --pretty

5 Create tomcat with 5 replicas on the default ingress network
  docker service create --name appserver -p 9999:8080 --replicas 5 tomcat

6 Perform a rolling network update from ingress to intelliqit2 network
  docker service update --network-add intelliqit2 appserver

7 Check if tomcat is now running on intelliqit2 network
  docker service inspect appserver --pretty

➔ **Note:** To remove from intelliqit2 network
     docker service update --network-rm intelliqit2 appserver

Docker Stack

- docker compose + docker swarm = docker stack
- docker compose + Kubernetes = kompose

- Docker compose when implemented at the level of docker swarm it is called docker stack. Using docker stack we can create an orchestreta a micro services architecture at the level of production servers

1 To create a stack from a compose file
 docker stack deploy -c compose_filename stack_name

2 To see the list of stacks created
 docker stack ls

3 To see on which nodes the stack services are running
 docker stack ps stack_name

4 To delete a stack
 docker stack rm stack_name

UseCase

➔ **Create a docker stack file to start 3 replicas of wordpress and one replica of mysql**

                         vim stack1.yml
```
---
version: '3.8'
services:
 db:
  image: "mysql:5"
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit

 wordpress:
  image: wordpress
  ports:
   - "8989:80"
  deploy:
   replicas: 3
```

➔ To start the stack file
   docker stack deploy -c stack1.yml mywordpress

➔ To see the services running
   docker service ls

➔ To check where the services are running
   docker stack ps mywordpress

➔ To delete the stack
   docker stack rm mywordpress

UseCase

➔ **Create a stack file to setup CI-cd architecture where a Jenkins container is linked with tomcats for qa and prod environments The jenkins containers should run only on Manager the qaserver tomcat should run only on Worker1 and prodserver tomcat should run only on worker2**

```
vim stack2.yml
---
version: '3.8'

services:
 myjenkins:
  image: jenkins/jenkins
  ports:
   - 5050:8080
  deploy:
   replicas: 2
   placement:
    constraints:
     - node.hostname == Manager

 qaserver:
  image: tomcat
  ports:
   - 6060:8080
  deploy:
   replicas: 3
   placement:
    constraints:
     - node.hostname == Worker1

 prodserver:
  image: tomcat
  ports:
   - 7070:8080
  deploy:
   replicas: 4
   placement:
    constraints:
     - node.hostname == Worker2
...
```

➔ To start the services
     docker deploy -c stack2.yml ci-cd

➔ To check the replicas
     docker stack ps ci-cd

UseCase

➔ **Create a stack file to setup the selenium hub and nodes architecture but also specify a upper limit on the h/w**

```
vim stack3.yml
---
version: '3.8'

services:
 hub:
  image: selenium/hub
  ports:
   - 4444:4444
  deploy:
   replicas: 2
   resources:
    limits:
     cpus: "0.1"
     memory: "300M"

 chrome:
  image: selenium/node-chrome-debug
  ports:
   - 5901:5900
  deploy:
   replicas: 3
   resources:
    limits:
     cpus: "0.01"
     memory: "100M"

 firefox:
  image: selenium/node-firefox-debug
  ports:
   - 5902:5900
  deploy:
   replicas: 3
   resources:
    limits:
     cpus: "0.01"
     memory: "100M"
```

➔ To deploy the stack
  docker stack deploy stack3.yml selenium

➔ To see the list of services
  docker service ls

➔ To check the hardware resources
  docker service inspect service_name --pretty

Global mode

➔ Certain containers have to be created one per server in the cluster then we use this global mode
➔ This ensures that one replica is created for every node as the node count increases the replicas count will also increase automatically

docker service create --name mynginx -p 8989:80 --mode-global nginx

➔ If we check the services no, we will see 3 replicas created as we have 3 servers in the cluster
   docker service ls

Docker Secrets

➔ This is a feature of docker swarm using which we can pass secret data to the services running in swarm cluster
➔ These secrets are created on the host machine and they will be available from all the replicas in the swarm cluster

1 Create a docker secret
  echo " Hello Intelliqit" | docker secret create mysecret -

2 Create a redis db with 5 replace and mount the secret
  docker service create --name myredis --replicas 5 --secret mysecret redis

3 Capture one of the replica container ids
  docker container ls

4 Check if the secret data is available
  docker exec -it container_id cat /run/secrets/mysecret