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 OS's as guest OS

The disadvantage of this approach is these applications running on the

guest OS have to pass through n number of layers 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 achive 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 stages of S/W development life cycle

Build---->Ship--->Run

=========================================================

Day 2-3

=========================================================

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

\*note: RHEL do not support Docker community edition. There we need to download enterprise edition

======================

Create an ubuntu linux machine using vagrant

==================================================

1 Download oracle virtual box from

https://www.virtualbox.org/wiki/Downloads

2 Install it

3 Download and install vagrant

https://www.vagrantup.com/downloads

4 Download the vagrant file and copy it into an empty folder

5 Open cmd prompt

6 Change directory to the folder where the vagrantfile is copied

cd path\_of\_folder

7 vagrant up

8 USername and password is:vagrant

========================================================================

Using AWS

================

1 Login in AWS account

2 Create ane new Ubuntu 20 instance

3 To connect to this ubuntu instance use gitbash

https://git-scm.com/downloads

=====================================================================\

Installing docker on Linux

================================

1 Open get.docker.com

2 Copy and paste 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 CLI of docker which accepts the docker commands

from the users and passes to a background process called docker deamon

Docker deamon: This accepts the commands coming from docker client

and routes them to work on docker images or container or the registry.

Docker registry: This is the location where docker images are stored

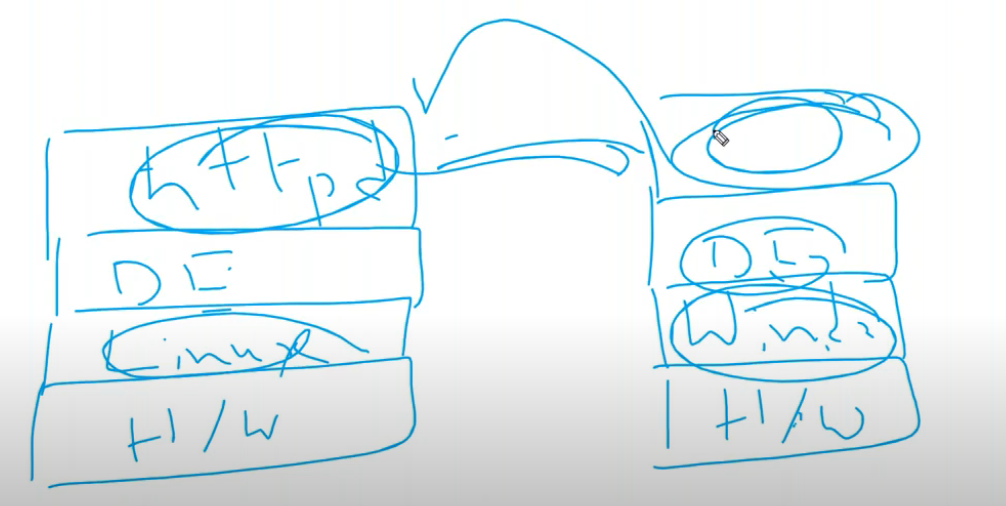
This if of 2 type

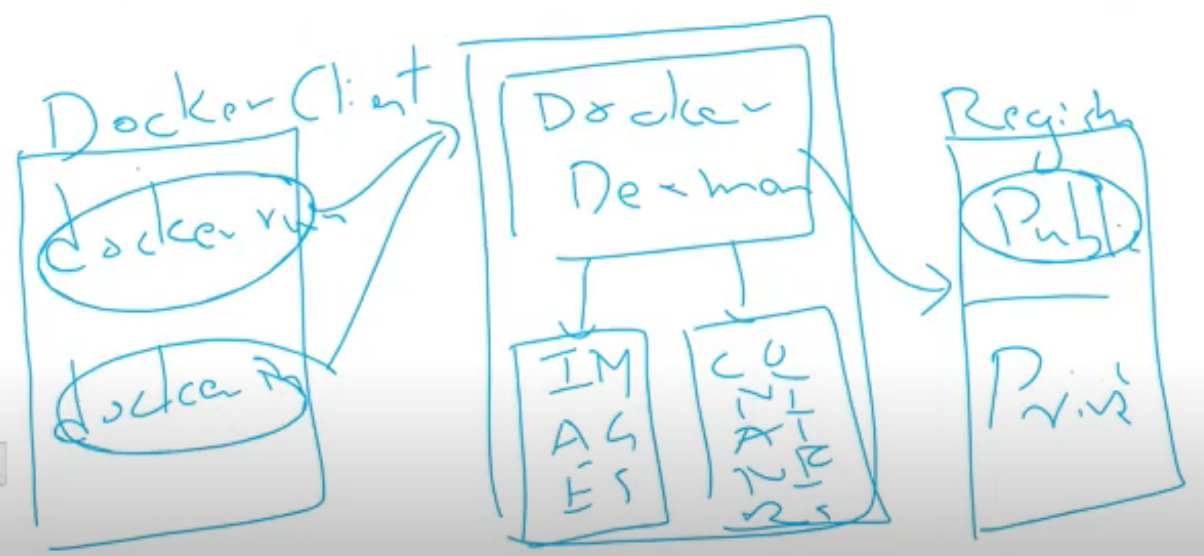
1 Public (hub.docker.com)

2 Private: This is set up on one of our internal servers

========================================================

Images that are designed to work on Linux will work on Windows BUT images that are designed to work on Windows itself will not work on Linux





Day 4

============================================================================

Important docker commands

===============================

Working on docker images

============================

1 To download a docker image

docker pull image\_name

2 To upload a docker image to docker registry

docker push image\_name

3 To search for a docker image on the registry

docker search image\_name

4 To see the list of images downloaded

docker image ls

(or)

docker images

5 To delete a docker image

docker rmi image\_name

6 To delete all docker images

docker system prune -af

7 To create a docker image from a customised container

docker commit container\_name/container\_id image\_name

8 To create an image from a docker file

docker build -t image\_name .

9 To get history for a docker image

docker image history image\_name

10 To get detailed info about a docker image

docker image inspect image\_name

Working on docker containers

=================================

11 To see the list of running containers

docker container ls

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

docker ps -a

13 To stop a running container

docker stop container\_name/container\_id

14 To start a stopped container

docker start container\_name/container\_id

15 To restart a container

docker restart container\_name/container\_id

To restart after 20 seconds

docker restart -t 20 container\_name/container\_id

16 To delete a stopped container

docker rm container\_name/container\_id

17 To delete a running container

docker rm -f container\_name/container\_id

18 To stop all running containers

docker stop $(docker ps -aq)

19 To delete all stopped containers

docker rm $(docker ps -aq)

20 To delete all containers (running and stopped)

docker rm -f $(docker ps -aq)

21 To get detailed info about a container

docker inspect container\_name/container\_id

22 To see the logs used by a container

docker logs container\_name/container\_id

23 To create a container

docker run image\_name

Run command options

-------------------

--name: Used to assign a name for the container

-it : Used to open interactive terminal in the container

-d : Used to run the containers in detached modes as a background process

-e : Used to pass environment variables to containers

-v : Used to attach a external fodler or device as a volume

--volumes-from : Used for sharing volumes between containers

-rm : USed to delete a container on exit

-p : Used for port mapping it will link the container port with

the host port

Eg: -p 8080:80 here 80 is the container port also called as internal

port and 8080 is host port also called as external port

-P: Used for automatic port mapping ie it will link the container port

with a host port that is greater than 30000

--link : This is used to create alink between multiple containers

to setup the microservices architecture.

--network: Used for creating a network of containers

-c: Used to specify the amount of maximum cpu that can be assigned

to a containers

-m : Used to sepcify the maximim amount of memory that a contaienr

can use

==================================================================

Day 5

=================================================================

24 To see the ports used by a container

docker port container\_name/container\_id

25 To come out of a container without exit

ctrl+p,ctlr+q

26 To go back into the container from which we have come out using above keystrokes

docker attach container\_name/container\_id

27 To run any command or application in a container from outside

docker exec -it container\_name/container\_id command

Eg: To start the bash shell in the container

docker exec -it container\_name/container\_id bash

===========================================================================

Working on docker networks

--------------------------

28 To see the list of availble docker networks

docker network ls

29 To create a docker network

docker network create --driver network\_type network\_name

30 To get detailed info about a network

docker network insepct network\_name/network\_id

31 To delete a network

docker network rm network\_name/network\_id

32 To connect a running container to a network

docker network connect network\_name/network\_id container\_name/container\_id

33 To disconnect a running container from a network

docker network disconnect network\_name/network\_id container\_name/container\_id

=========================================================================

Working on docker volumes

=============================

35 To see the list of docker volumes

docker volume ls

36 To create a volume

docker volume create volume\_name

37 To delete a volume

docker volume rm volume\_name/volume\_id

38 To get detailed info about a volume

docker volume inspect volume\_name/volume\_id

=============================================================================

Day 6

=============================================================================

==============================================================================

UseCase 1

===============

Create an nginx container in detached mode

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

To check if the nginx contianer is running

docker container ls

To access the ngonx from browser

public\_ip\_dockerhost:8888

=============================================================================

UseCase 2

===================

Create jenkins container and do automtic port mapping

docker run --name jenkins -d -P Jenkins/jenkins

To see the ports used by jenkins

docker port jenkins

To access the jenkins from browser

public\_ip\_of\_dockerhost:port\_no\_from\_previous\_step

===================================================================================

UseCase 3

Start centos as a container and launch interactive terminal on it

docker run --name c1 -it centos

=============================================================================

Day 4

==============================================================================

UseCase 1

Create a mysql container and go into its bash shell

Login as mysql root user and create few tables.

1 Create a mysql container

docker run --name db -d -e MYSQL\_ROOT\_PASSWORD=intelliqit mysql:5

2 To open interactive bash shell in the container

docker exec -it db bash

3 To login into the db as root user

mysql -u root -p

Enter password "intelliqit"

4 To see the list of availble databases

show databases;

5 To move into any of the above database

use db\_name;

Eg: use sys;

6 To create emp and dept tables here

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

Copy the code from emp and dept tables and paste in the myswl container

7 To check if the emp and dept tables are created

select \* from emp;

select \* from dept;

============================================================================

Linking of Container

==========================

To create a multi container architecture we have use the following ways

1 --link option

2 Docker compose

3 Docker network

4 Python Scripts

5 Ansible playbooks

UseCase

Create 2 busybooks 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 contaienr 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

==============================================================================

UseCase

Create a mysql container and link it with wordpress container

1 Create a mysql container

docker run --name db -d -e MYSQL\_ROOT\_PASSWORD=intelliqit mysql:5

2 To create a wordpress container and link with mysql container

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

3 Check if the mysql and wordpress containers are running

docker container ls

4 To acces wordpress from browser

public\_ip\_dockerhost:8888

=========================================================================

Day 7

=========================================================================

UseCase

================

Create a jenkins container and link with 2 tomcat containers

one for QAserver and another for prodserver

1 Create a jenkins container

docker run --name jenkins -d -p 5050:8080 jenkins

2 To access jenkins from browser

public\_ip\_of\_dockerhost:5050

3 Create a tomcat container as qaserver and link with jenkins

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

4 Create another tomcat container as prodserver and link with jenkins

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

======================================================================

Use Case

================

Setup a postgres db and link with adminer container to access db from browser

1 Create a postgres db container

docker run --name db -d -e POSTGRES\_DB=intelliqit -e POSTGRES\_USER=myuser -e POSTGRES\_DB=mydb postgres

2 Create an adminer container and link with postgres

docker run --name myadminer -d -p 9999:80 --link db:postgres adminer

3 To access from level of browser

public\_ip\_of\_dockerhost:9999

==============================================================================

UseCase

===========

Setup LAMP architecture where a mysql container can be linked

with an apache and php container

1 Create a mysql container

docker run --name db -d -e MYSQL\_ROOT\_PASSWORD=intelliqit mysql

2 Create an apache and link with mysql container

docker run --name apache -d -p 9090:80 --link db:mysql httpd

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

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

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

docker inspect php

Search for "Links" section

==============================================================================

UseCase

=================

Create a testing environment where a selenium hub container

should be linked with 2 node containers one with chrome

installed and other with firefox installed. The testers should be

able to run the cross browser, cross platform automation

test scripts

1 Create a selenium hub image

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

2 Create a chrome node and link with the hub container

docker run --name chrome -d -p 5901:5900 --link hub:selenium

selenium/node-chrome-debug

3 Create a firefox node and link with the hub container

docker run --name firefox -d -p 5902:5900 --link hub:selenium

selenium/node-firefox-debug

4 Check if all 3 containers are running

docker container ls

5 The above 2 containers are GUI containers and to access the GUI of

these containers

a) Install VNC viewer from https://www.realvnc.com/en/connect/download/viewer/

b) Open vnc viewer

c) public\_ip\_of\_dockerhost:5901 and 5902

d) Click on continue--->Enter password:secret

==============================================================================

Day 8

===============================================================================

Docker Compose

=====================

This is used for creating a multi container architecture which is reusable.

Docker compose uses yml files

Install docker-compose

============================

1 Open https://docs.docker.com/compose/install/

2 Click on Linux tab

3 Copy and paste the commands

4 To check the version of docker compose

docker-compose --version

==========================================================================

UseCase

Create a docker compose file to setup a mysql container linked with

a database container

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 the containers

docker-compose stop

To stop and delete the containers

docker-compose down

===================================================================

UseCase

Create a docker compose file to setup ci-cd environment

where a jenkins is linked with 2 tomcat containers

vim docker-compose.yml

---

version: '3.8'

services:

myjenkins:

image: jenkins/jenkins

ports:

- 5050:8080

container\_name: myjenkins

qaserver:

image: tomcat

ports:

- 6060:8080

links:

- myjenkins:jenkins

container\_name: qaserver

prodserver:

image: tomcat

ports:

- 7070:8080

links:

- myjenkins:jenkins

container\_name: prodserver

...

===========================================================================

UseCase

================

Create a docker compose file to setup the LAMP architecture

---

version: '3.8'

services:

mydb:

image: mysql

environment:

MYSQL\_ROOT\_PASSWORD: intelliqit

container\_name: mydb

apache:

image: httpd

ports:

- 9090:80

container\_name: apache

links:

- mydb:mysql

php:

image: php:7.2-apache

links:

- mydb:mysql

- apache:httpd

container\_name: php

...

==========================================================================

Day 9

========================================================================

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 deleted all the

data of the container will be 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 is deleted.

UsedCase

============

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 c1 -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

ctlr+p,ctlr+q

4 Create another centos container c2 and it should used 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

ctlr+p,ctlr+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

ctlr+p,ctlr+q

10 Go into any of the 3 contianers 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"

ls

=====================================================================

Day 10

=====================================================================

Docker volume containers

----------------------------

These volumes are bi-directoinal 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 customsied

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 installtion 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 downlaod files from some remote server

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

VOLUME : Used for automatic volume mounting ie we will have a volume

mounted automatically when the container start

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

===========================================================================

Day 11

=============================================================================

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:

vim dockerfile

FROM ubuntu

RUN apt-get update

RUN apt-get install -y git

docker build -t myubuntu .

If we build an image from the above dockerfile docker saves all

these instructions in the dockercache 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

one more time without reading from cache we use "cache busting"

docker build --no-cache -t myubuntu .