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

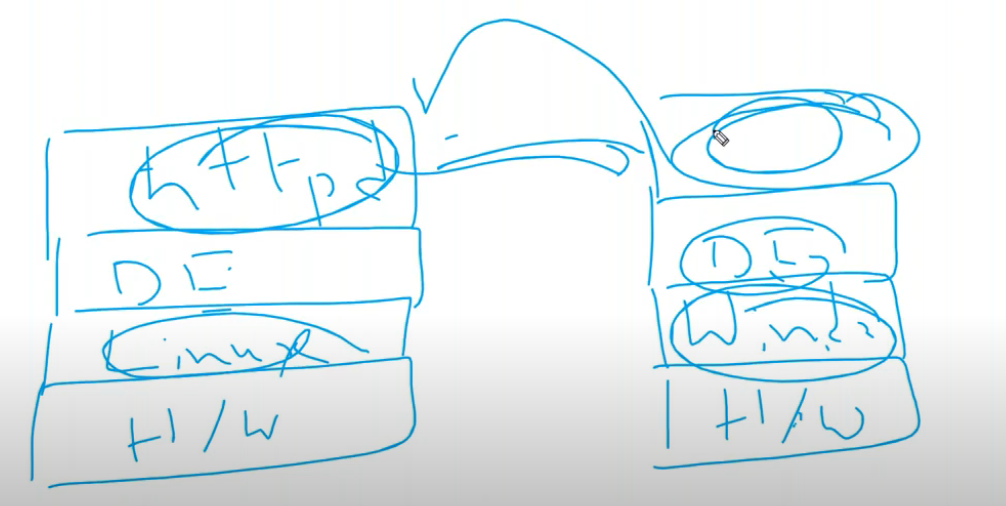
This if of 2 type

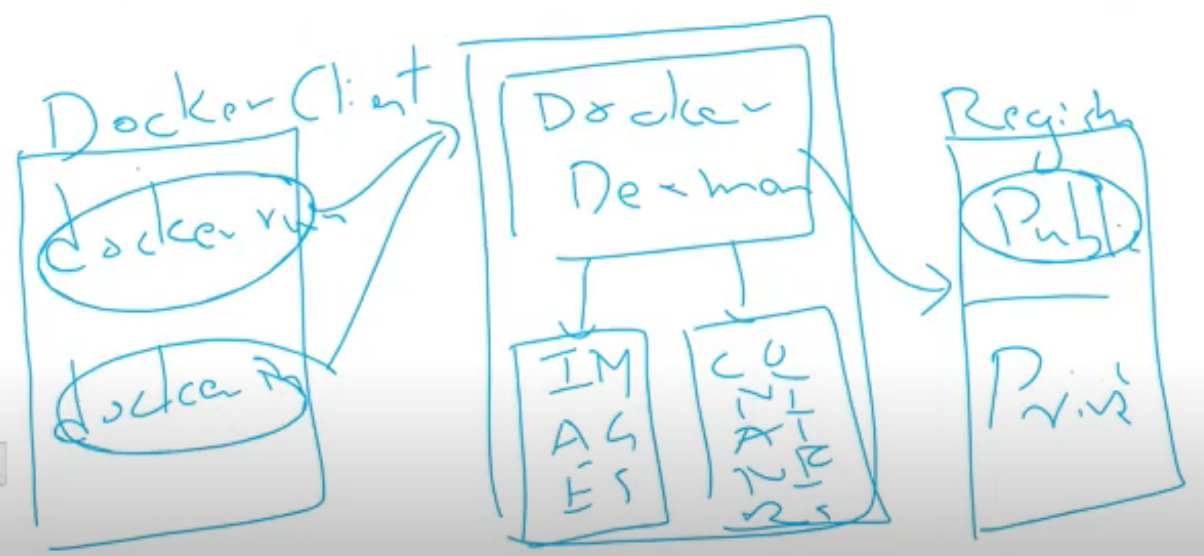
1 Public (hub.docker.com)

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\_PASSWORD=mydb postgres

OR

$ docker run --name some-postgres -e POSTGRES\_PASSWORD=mysecretpassword -d postgres

OR

docker run --name mydb -e POSTGRES\_DB=intelliqit -e POSTGRES\_USER=intelliqit -e POSTGRES\_PASSWORD=mysecretpassword -d 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 .

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

Day 12

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

Create a shell script to install multiple s/w's and copy this

into the docker image and execute it a the the time os creating the image

1 Create the shell script

vim script.sh

apt-get update

for x in tree git

do

apt-get install -y $x

done

2 Give excute permissions on that file

chmod u+x script.sh

3 Create the dockerfile

vim dockerfile

FROM ubuntu

MAINTIANER intelliqit

COPY ./script.sh /

RUN ./script.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 script.sh is present in / and also see if tree and git are installed

ls /

git --version

tree

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

UseCase

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

Create a dockerfile from ubuntu base image and install ansible in it

1 Create a dockerfile

FROM ubuntu

MAINTAINER intelliqit

RUN apt-get update

RUN apt-get install -y software-properties-common

RUN apt-get install -y ansible

2 Create an image from the above file

docker build -t ansible .

3 Create a container from the above image and check if ansible is present

docker run --name a1 -it ansible

ansible --version

\*note: suppose you created Mysql database container and then wordpress container linked with Mysql container. Now if you ping Mysql db container from wordpress it will pinged but if you try to ping wordpress from Mysql, it won’t get pinged

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

Day 13

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

Create a dockerfile from ubuntu base image and make

/data as the default volume

1 Create a dockerfile

vim dockerfile

FROM ubuntu

MAINTAINER intelliqit

VOLUME /data

2 Create an image from the above dockerfile

docker build -t myubuntu .

3 Create a container from the above image and check for the volume

docker run --name u1 -it myubuntu

ls (we should see the data folder)

4 Go into the data folder and create some files

cd data

touch file1 file2

exit

5 Check for the mount section and copy the source path

6 Delete the container

docker rm -f u1

7 Check if the files are still present

cd "source path"

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

Create a docker file from nginx base image and expose 80 port

1 vim dockerfile

FROM nginx

MAINTAINER intelliqit

EXPOSE 90

2 Create an image

docker build -t mynginx .

3 Create a container from above image

docker run --name n1 -d -P mynginx

4 Check for the ports exposed

docker port n1

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

Create a dockerfile from ubuntu base image and download jenkins.war

into it

1 Create a dockerfile

vim dockerfile

FROM ubuntu

MAINTAINER 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

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

Day 14

When we create number of containers and run command 🡪

# docker container ls

We get different process for each container in command section.

If command is “/bin/bash” then we can enter in interactive terminal directly while for others we need give command🡪 # docker exec….

Create a dcokerfile from ubuntu base image and install

java in it, download jenkins.war and make

"java -jar jenkins.war" as the default process of the

container

1 vim dockerfile

FROM ubuntu

MAINTAINER intelliqit

RUN apt-get update

RUN apt-get install -y openjdk-8-jdk

ADD https://get.jenkins.io/war-stable/2.263.4/jenkins.war /

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

2 Create an image from the above file

docker build -t myubuntu .

3 Create a container from the above image and we will see that

it behaves like a jenkins container

docker run --name u1 -it myubuntu

4 Check the default process that is running

docker container ls

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

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

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

UseCase

Create a dockerfile fromj centos base image and install

httpd in it make httpd as the default process

1 Create the index.html

<html>

<body>

<h1>Welcome to IntelliQIT</h1>

</body>

</html>

2 Create the dockerfile

vim dockerfile

FROM centos

MAINTAINER intelliqit

RUN yum -y update

RUN yum -y install httpd

COPY index.html /var/www/html

ENTRYPOINT ["/usr/sbin/httpd","-D","FOREGROUND"]

EXPOSE 80

3 Create an image from the above dockerfile

docker build -t mycentos .

4 Create a container from the above image

docker run --name c1 -d -P mycentos

5 Check the ports used by container

docker container ls

6 To access the from browser

public\_ip\_of\_dockerhost:port\_from\_step5

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

\*note: in linux when run command 🡪 # which nginx

Instead of nginx you can give anything

o/p will show a location where its installed

Day 15

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

CMD and ENTRYPOIT

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

Both are 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 otherhand 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"

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

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 ipaddress 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 ipaddress 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 ipaddress 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

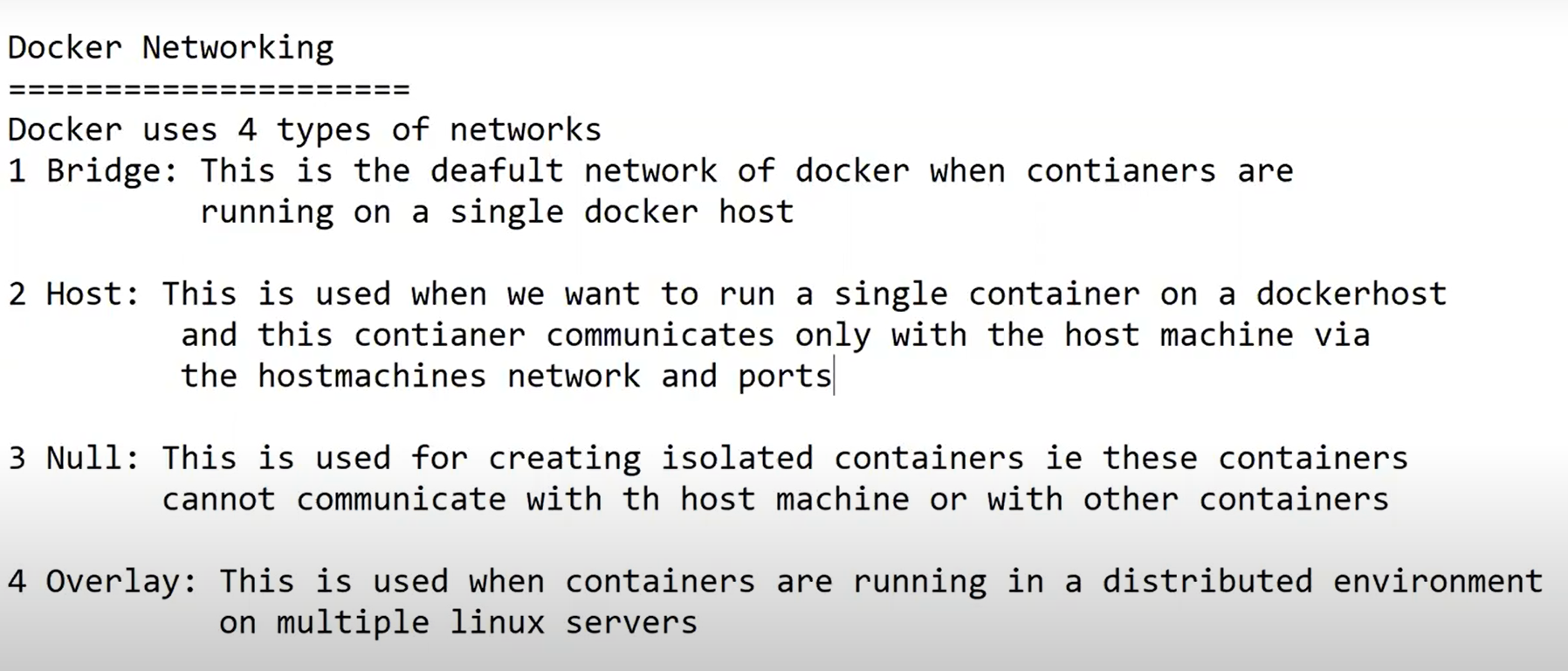
11 But c1 and c3 should not ping each other

docker attach c3

ping ipaddress\_of\_c1 (It should not ping)

Note: To create network with a specific subnet range

docker network create --driver bridge --subnet=192.168.2.0/24 intelliqit3



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

Day 16

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

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

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

UseCase

Create a docker compose file which starts mysql and wordpress as

containers also this compose file should create 2 networks

abc,xyz. On abc network it should run the mysql container

and xyz network it should run the wordpress container

vim docker-compose.yml

version: '3.8'

services:

db:

image: mysql:5

environment:

MYSQL\_ROOT\_PASSWORD: intelliqit

networks:

- abc

wordpress:

image: wordpress

ports:

- 9090:80

networks:

- xyz

networks:

abc: {}

xyz: {}

...

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

UseCase

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

Create a docker compose file to start mysql and wordpress as container

it should also create 2 volumes one for wordpress and other for mysql

vim docker-compose.yml

version: '3.8'

services:

db:

image: mysql:5

environment:

MYSQL\_ROOT\_PASSWORD: intelliqit

volumes:

- db:/var/lib/mysql 🡨 backup will be taken from this path into volume- db

wordpress:

image: wordpress

ports:

- 9090:80

volumes:

- wordpress:/var/www/html🡨 backup will be taken from this path into volume- wordpress

volumes:

db:

wordpress:

To create containers from the above file

docker-compose up -d

Check if 2 new volumes are created

docker volume ls

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

UseCase

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

Create a dockerfile to create a customised jenkins image

and this dockerfile should be built as image from the docker composefile

vim dockerfile

FROM jenkins/jenkins

MAINTAINER intelliqit

USER root

RUN apt-get update

RUN apt-get install -y git maven

vim docker-compose.yml

---

version: '3.8'

services:

jenkins:

build: .

mytomcat:

image: tomcat

ports:

- 8080:8080

...

To create containers from the above file

docker-compose up -d

Check if a new jenkins image has be created

docker images

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

Working on docker registry

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

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 centos image

a) Create a centos container and install git init

docker run --name c1 -it centos

yum -y update

yum -y install git

exit

b) Save this container as an image

docker commit c1 intelliqit/mycentos

3 Login into dockerhub

docker login

Enter username and password of dockerhub

4 Push the customised image

docker push intelliqit/mycentos

Day 17

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

Private Registry

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

This can be created using a docker image called as "registry"

We can start this as a container and it will allow us to

push images into the registry

1 Create registry as a container

docker run --name lr -d -p 5000:5000 registry

2 Download an alpine image

docker pull alipne

3 Tag the alpine with the local registry

docker tag alpine localhost:5000/alpine

4 Push the image to local registry

docker push localhost:5000/alpine

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

Container Orchestration

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

This is the process of handling docker containers running on multiple linux servers in a distributed environment

Advantages

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

1 Load Balancing

2 Scaling

3 Rolling update

4 High Availability and Disaster recovery(DR)

LoadBalancing

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

Each container is capable of sustaining a specific user load.

We can increase this capacity by running the same application on multiple containers(replicas)

Scaling

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

We should be able to increase or decrease the number of containers on which our applications are running without the end user experiencing any downtime.

Rolling update

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

Application running in a live environment should be upgraded or downgraded to a different version without the end user having any downtime

Disaster Recovery

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

In case of network failures or server crashes still the container orchestration tools maintain the desired count of containers and thereby provide the same service to the end user

Popular container orchestration tools

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

1 Docker Swarm

2 Kubernetes

3 OpenShift

4 Mesos

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

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 token-id

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).

\*note🡪 next day if you want to add one more ec2 as a worker node generate token by running following command

# docker swarm join-token worker

Day 18

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

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

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

Scalling

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

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 tomcat with 4 replicas and scale it to 8 and scale it down to 2

1 Create tomcat with 4 replicas

docker service create --name appserver -p 9090:8080 --replicas 4 tomcat

2 Check if 4 replicas are running

docker service ps appserver

3 Increase the replicas count to 8

docker service scale appserver=8

4 Check if 8 replicas are running

docker service ps appserver

5 Decrese the replicas count to 2

docker service scale appserver=2

6 Check if 2 replicas are running

docker service ps appserver

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

Rolling updates

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

Services running in docker swarm should be updated from one 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 its place redis:4 replicas are running

docker service ps myredis

docker service ps myredis | grep \\_ 🡨 shutting down containers will see

docker service ps myredis | grep -v \\_ 🡨 reverse grep operation

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

\*note: lets say redis 2 was running and you upgrade it to redis 3 and then to redis 4. Now if you rollback, it will be redis 3. Now again if you rollback then it will be redis 4 and not redis 2

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

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 tokenid for a machine to join swarm as worker

docker swarm join-token worker

To generate the tokenid 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

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

Day 19

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

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 9090:80 --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 is also responsible for maintaining the minimum number of managers.

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

Min count of manager required > half of the total count of Managers

Total Manager Count - Min Manager Required

1 - 1

2 - 2

3 - 2

4 - 3

5 - 3

6 - 4

7 - 4

Usedcase🡪 when manager gets failed

1. Launch 2 ec2 and name them as manager1 and manager2
2. Install docker in both
3. Change name in hostfile
4. Generate token in manager and join newly created managers in that cluster

# docker swarm join-token manager

1. # docker service create --name webserver --replicas 5 -p 8888:80 nginx
2. Check

# docker service ps webserver

1. Since we have 3 managers minimum managers should be 2
2. Login to manager3 and run

# docker swarm leave --force

--force is because we are working on manager machine. For worker node its not needed

This will leave the cluster

1. Login to previous manager and check

# docker node ls

Manager3 must be down

1. # docker service ps webserver

Container running on manager3 will be migrated somewhere here

1. Down one more manager and check

Docker sworm will not work. No commands will work though containers in managers (which we downed) will be there and working fine.

Day 20

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

Overlay Networking

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

This is the deafult network used by swarm

and this network perfrom network load balancin

ie even if a service is running on a specicfic worker we can

access if from orther slave

UseCase

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

Start nginx with 2 repliacs and check if we can acces it from

browser from manager and all workers

1 Create nginx

docker service create --name webserver -p 8888:80 --replicas 2 nginx

2 Check where these 2 replcas are running

docker service ps webserver

These repliacs will be running on only 2 nodes and we will have a third

node where it it not running

3 Check if we can access nginx from the third node where it is not present

public\_ip\_of\_thirdnode:8888

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

UseCase

===========

Create 2 overlay networks intelliqit1 and intelliqit2

Create httpd with 5 replacs on intelliqit1 network

Create tomcat with 5 replicas on default overlay "ingres" 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 replcias 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 fromat

docker service inspect webserver --pretty

5 Create tomcat with 5 replicas on the deafult ingres network

docker service create --name appserver -p 9999:8080 --replicas 5 tomcat

6 Perform a rolling network update from ingres 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 serives are running

docker stack ps mywordpress

To delete the stack

docker stack rm mywordpress

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

Day 21

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

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 contianers 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"

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

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

availbale from all the replicas in the swarm cluster

1 Create a dcoker 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 contianer id

docker container ls

4 Check if the secret data is available

docker exec -it container\_id cat /run/secrets/mysecret

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

Create 3 secrets for postgres user,password and db

and pass them to the stack file

1 Create secrets

echo "intelliqit" | docker secret create pg\_password -

echo "myuser" | docker secret create pg\_user -

echo "mydb" | docker secret create pg\_db -

2 Check if the secrets are created

docker secret ls

3 Create the docker stack file to work on these secrets

vim stack6.yml

---

version: '3.1'

services:

db:

image: postgres

environment:

POSTGRES\_PASSWORD\_FILE: /run/secrets/pg\_password

POSTGRES\_USER\_FILE: /run/secrets/pg\_user

POSTGRES\_DB\_FILE: /run/secrets/pg\_db

secrets:

- pg\_password

- pg\_user

- pg\_db

adminer:

image: adminer

restart: always

ports:

- 8080:8080

deploy:

replicas: 2

secrets:

pg\_password:

external: true

pg\_user:

external: true

pg\_db:

external: true

...

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