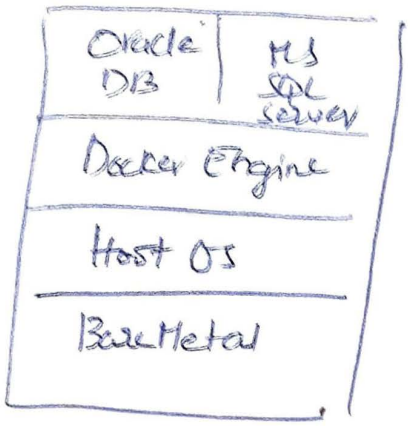
25/02/20 1st session DOCKER

Virtualization: Here we have a bare metal (H/W) on top of which we install the host OS and on the host OS we install an application called hypervisor (VMware Esxi, Citrix Xen, Microsoft Hyper V) and on the hypervisor we can install any guest os on the this guest os we install the applications that we want - The disadvantage is these applications have to pass through so many layers to access H/W resources

Containerization: Here we have a baremetal on top of which we install the host OS and on the host OS we install an application called "Docker Engine" On the Docker Engine we can run any application in the form of Containers. These containers pass through less no of layers to access the H/W resources so they are much faster.



Docker performs "process isolation" i.e it removes the dependency that an application has on an underlying OS and allows it to run directly on top of docker Engine. Due to this reason it is more cost effective as we need not spend money on the licensing of the OS

* Docker containers can be used at all the stages of SDLC

Build ---> Ship ---> Run

* Docker comes in 2 flavours

① Docker CE (Community Edition)
② Docker EE (Enterprise Edition)

Installing Docker on Windows

① ~~open~~ docker download (Google)
② Download docker and Install it

Note: docker can be installed only on windows 10 prof 64 bit version or windows 2016 Server Edition

Note: Docker on windows activates an application called Hyper V once this Hyper V is activated it will not allow any other Virtualization s/w to run

Docker Session 2 (26-02-20):

get.docker.com

creating a Ubuntu Linux Server on AWS

1 Signup for a free AWS account
2 Sign into that account
3 Click on Services ---> Click on EC2 ---> Click on Instances
4 Click on Launch Instance

Search for Ubuntu server 18 Click on Select

Click on configure Instance details

Click on Add Storage

Click on Add tags

Click on Configure Security Groups.

Click on Add Rule Click on All traffic Click on Anywhere (Source)

Click on Review and Launch

Click on Launch

~~Ste~~ Select create a new key pair

Enter some keypair Name, Click on download Keypair

Click on Launch Instances, Click on View Instances

To connect to this remote server linux download and install Git
https://git-scm.com/downloads

Install Docker on Linux server

1 Open get.docker.com

2 Copy the below 2 command and paste in linux terminal

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

To Check the version of docker

```
docker --version
```

Images and containers:

→ A Docker is a combination of binaries and Libraries which are necessary for a software application to run.

→ A running instance of an image is known as Container

Any no. of containers can be created from one Docker Image.

Docker Host:

The Server (Windows/Linux) where Docker Engine is installed is known as the Docker host

Docker Client:

This is a process which runs in the Docker Engine and it is responsible for accepting the docker commands from the user. and pass it to a background process called Docker Deamon.

Docker Daemon: This is responsible for accepting the Docker commands from the Docker Client and depending on the kind of command it will route them to work on Images Containers or the Registry

Registry: Registry is the repository where Docker Images are stored. This registry is of 2 types Public and Private. Public is hub.docker.com and it is maintained by Docker Corporation. Private registry is setup within our organisation servers and only our team members can access these images.

Docker Session 3 (27-02-20)

Important Docker Commands:

1 Working on docker image

```
docker pull image_name
```

2. To push an image into docker hub

```
docker push image_name
```

3 To search for an image

```
docker search image_name
```

4 To see the list of images down-loaded into docker host

```
docker image ls
    or
docker images
```

5. To delete a docker image

```
docker rmi image_name/image_id
```

6. To delete all the unused docker images

```
docker system prune -a
```

7. To create a docker image from a docker file

```
docker build -t image_name
```

8. To create a docker image from a customised container

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

9. To get detailed info about a docker image

```
docker inspect image_name/image_id
```

Working on docker containers:

10. To see the list of running containers

```
docker container ls
```

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

```
docker ps -a
```

12. To start a stopped container

```
docker start container_name/container_id
```

13 To stop a running container

```
docker stop container_name/container_id
```

14 To restart a running container

```
docker restart container_name/container_id
```

To restart a container after 10 seconds

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

15. To delete a running container

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

16. To delete a stopped container

docker rm container_name/container_id

17. To stop all running containers

docker stop $(docker ps -aq)

18. To delete all the stopped containers

docker rm $(docker ps -aq)

19 To delete all containers (running and stopped)

docker rm -f $(docker ps -aq)

20 To get detailed info about a container

docker inspect container_name/ container_id

21 To see the logs generated by a container

docker logs container_name/ container_id

22 To See the ports used by a container

docker port container_name/ container_id

23 To come out of a container without Exit

Ctrl+P, Ctrl+q

24 To get back into the same container again

docker attach container_name / container_id

25. To run a process or application in container

docker exec -it container_name/ container_id bash

26. To Create Container

docker run Emaage_name

run command options

--name : Used for giving a name to a container

-d : Used to run a container in detached mode (background)

-it : Used for opening interactive terminal in a container

-v : Used for attaching an external device or directory as a volume.

--volumes_from : Used for sharing volumes between multiple containers.

--link : Used to link docker containers to create a ~~multi~~ multi container architecture

-p : Used for port mapping. It will link the container port (Internal port) with docker hostport (external port)

eg: -p 8080:80 Here 8080 will be docker hostport and 80 is container port

-P : Used for automatic port mapping. ie the container port will be mapped with the host port which will be automatically allocated and this host port will be always greater than 30000.

-rm : Used to delete a container on exit from the container

-e : Used to pass enviroment variables to the container

-m : Used for specifying how much maximum memory a container can use

-c : Used for allocating

--network : Used to specify on which network a container should run

Working on docker network

27 To see all the docker networks

```
docker network ls
```

28 To create a new docker network

docker network create --driver network_type network_name

29 To get detailed info about a network

docker network inspect

30 To delete a network

docker network rm network_name/network_id

31 To attach a running container to a network

docker network connect network_name/network_id
container_name/container_id

32. To disconnect a container from an network

docker network disconnect network_name/network_id
Container_name/ container_id

Working on docker Volumes:

33. To see the list of docker volumes

docker volume ls

34 To get detailed info about a volume

docker volume inspect volume_name/ volume_id

35 To create a volume

docker volume create volume_name

Docker Session 4 (28.02.20)

Use case 1

Start nginx as a container and perform port mapping

Check if we can access the homepage of nginx

① start nginx as a container

docker run --name webserver -d -p 9999:80 nginx

② To check if the nginx container is running

docker container ls

③ To access the nginx container from a browser

Use case 2

start httpd as a container and perform automatic port mapping

① Start httpd as a container

docker run --name appserver -d -P httpd

② To check if the httpd container is running

docker container ls

capture the host port from this output

③ To access the container from a browser

Launch any browser

public_ip_of_dockerhost: host_port_captured_from_step_2

Docker Session 5 (29-02-20)

* Create an ubuntu OS container and open an interactive terminal in the container.

① Create an ubuntu container

docker run --name u1 -it ubuntu

② To come out of the ubuntu container without exit

Ctrl + p, Ctrl + q

③ To see the list of running containers

docker container ls

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

① Create an mysql container

docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=intelliq
mysql:5

② To open interactive terminal in the mysql container

docker exec -it mydb bash

③ To login as a root user

mysql -u root -p

Enter password: Intelliq

④ To see the list of databases

Show databases;

⑤ To move into any of the above database

use sys;

⑥ To create Emp and dept tables

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

Copy the code for creating tables and paste in the mysql docker conta

⑦ To see the tables data

```
Select * from Emp;
Select * from dept;
```

Creating Multi container Architecture:

Multiple docker containers can be linked with each other to create a multi container architecture. This can be done by the following ways

1. Using --link option
2. Using docker compose
3. Docker networking
4. Python Scripts

--link option: This is a docker run command option and it is depricated

Usecase 1

Start two busybox containers C1 and C2 and create a link between both of them. Check wether we can ping from one container to another

① Create a busybox container 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 the c1 container

docker run --name c2 -it --link c1:my_c1 busybox

4 In the c2 container check if we can ping to c1

Ping c1

Use case 2 Create a development environment where a wordpress container should be linked with a mysql container

① Create a mysql container

docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=intell
mysql:5

② Create a wordpress container and link it with the mysql container

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

③ To access the homepage of the wordpress container

Launch any browser

public_ip_of_dockerhost : 8888

Docker Session 6 (21.03.20)

Create a Jenkins container and link it with 2 tomcat containers. One for QA server another for Prod server.

① start jenkins as a container

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

② To access the homepage of jenkins

Launch any browser

public_ip_of_dockerhost:5050

③ start tomcat as a container (QA server) and link with jenkins

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

④ start another tomcat as a container (Prod server) and link with jenkins

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

⑤ To check which containers are linked

docker inspect container_name

Go to "Links" section and check the data

Create an Environment where Ghost application can be linked with Mysql database (Ghost is a blogging software)

① start mysql as a container

docker run --name mydb -d -e MYSQL_ROOT_PASSWORD:intellig mysql:5

② start ghost as a container and link with mysql container

docker run --name ghost -d -p 8888:2368 --link mydb:mysql ghost

③ To access the ghost application

Launch any browser

Public_ip_of_dockerhost : 8888

Create a LAMP architecture where mysql container should be linked with Apache and php containers

① Create mysql Container

docker run --name mydb -e MYSQL_ROOT_PASSWORD=Intellig mysql

② start an apache container and link with mysql container

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

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

docker run --name php --link mydb:mysql --link apache:httpd

-d php:7.2-apache

④ To Check running containers

docker container ls

→ Create a testing Environment where a selenium hub container will be linked with 2 node containers. One with firefox and another with Chrome. A selenium automation Tester should now be able to run his cross browser automation test programs in this Environment.

① start selenium hub as a Container

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

② start a Chrome node as a container and link with hub Container

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

selenium/node-chrome-debug

③ Start a firefox node as a container and link with hub container

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

④ The above 2 firefox and Chrome are Ubuntu GUI Containers

To access the GUI of these Containers

Install VNC viewer and Launch

public_ip_of_dockerhost:5901 (or) 5902

Click continue

password: secret

Docker Session 7 (03-03-20):

Docker Compose This is used for creating a multi container architecture. The --link option of docker run command is depricated.

Docker compose files are created using YAML and they are more reusable.

Installing Docker Compose:

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

② Click on Linux tab and copy paste the commands

To check the version: docker-compose --version

Sample YAML File:

```
---
Intelliq:
trainers:
  devops: Sai
  aws: Shesh
Coordinators:
  devops: Lakshmi
  aws: Shailaja
...
```

Create a docker compose file for linking MySQL container with wordpress container.

① Vim docker-compose.yml

```
---
Version: '3'

Services:
 mydb:
  image: mysql:5
  environment
  * MYSQL_ROOT_PASSWORD: intelliq

 mywordpress
  image: wordpress
  ports:
   - 8888:80

  Links:
   - mydb:mysql

...
```

② To start the environment

docker-compose up

To start enviroment in detached mode

docker-compose up -d

③ To stop the environment

docker-compose stop

④ To delete the environment

docker-compose down

Create a dockercompose file for linking a jenkins container with 2 tomcat containes. for a devops engineer to perform CI/CD.

① Vim docker-compose.yml

```
---
Version : '3'

services :
 jenkins:
   image : jenkins
   ports :
   - 5050 : 8080

 qaserver :
   image : tomcat
   ports :
    - 6060 : 8080

   links :
    - jenkins : my jenkins

 prodserver
   image : tomcat
   ports :
     - 7070 : 8080

   links :
     - jenkins : myjenkins

...
```

Docker Session 6 (04-03-20):

→ Create a docker compose file to link a mysql container with ghost container.

Vim docker-compose.yml

```
---
Version: '3'

Services:
 mydb:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWOR= Intelliq

 ghost:
  imag: ghost
  ports:
  - 8888:2368

  links:
  - mydb:mysql
...
```

→ Create a docker compare file for setting up the LAMP architecture

Vim abc.yml

```
---
Version: '3'
Services:
 mydb:
  image: mySql
  environment:
   MYSQL_ROOT_PASSWORD= Intelliq

 apache:
  image: httpd
  ports:
   -9999:80
  links:
  -mydb:mysql

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

#date

Run the above file

docker-compose -f abc.yml up -d

→ Create a docker compose file for setting the testing Environment where a selenium hub container should be linked with 2 Node containers. One with firefox and another with chrome.

```
Vim docker-compose.yml
---
version: '3'

services:
  hub:
    image: selenium/hub
    ports:
      - 4444:4444

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

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

# docker-compose up
```

Volumes:

Docker containers are ~~ephemeral~~ ephemeral (temperary) where as data proces by the container should be persistent. Generally whenever a container is deleted, data in the container also will be lost.

To preserve the data we can use volumes.

A ~~volume~~ docker volume is an external device or directory mounted on a container in such a way that even after the container is deleted data will still be available on the host machine.

Docker uses 3 types of volumes

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

① Simple docker volume: This is used only for preserving the data even after the container is deleted. This data ~~is~~ will be available on the host machine. But it cannot be used by other containers.

Use case: Create a directory /data and mount it on Centos Container.

Create ~~some~~ some files in this mounted directory in the container. Delete the container and check if these files are still available on the host machine.

① Create a directory

mkdir /data

② Create a centos container and mount /data as a volume on it

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

③ In the Centos c1 container go into data folder and Create few files

cd /data

touch file1 file2

exit

④ Identify the mounted location of the volume

docker inspect c1

Go to "Mounts" section and copy the "source" path

⑤ Delete the container

docker rm -f c1

⑥ Check if the data is still present

cd "source_path_copied_in_step4"

ls

05-03-20 – Class Cancelled

Docker Session 8 (06-03-20):

Today it's not a regular session as there is some technical problem

Discussed about Interview perspective terminology

Video will be shared.

Docker Session 9 (07-03-20):

Sharable docker Volumes: When we want multiple containers to share a volume between themselves we can use this feature.

Use Case: Create a directory /data

Create 3 centOs Containers C1 C2 and C3

On C1 Mount /data as a volume. C2 Container should use the volume used by C1. C3 container should use the Volume used by C2. Later delete all the 3 containers and Check if the data is still available on the host machine.

① Create /data folder

mkdir /data

② Start centos as a container and mount /data as a volume

docker run --name C1 -it -v /data centos

③ In the Centos C1 container go into the volume and create files

cd /data
touch file1 file2

Come out of the container without exit (Ctrl+p, Ctrl+q)

④ Create another Centos container C2 and this container should use the Volume used by C1

docker run --name C2 -it --volumes-from C1 centos

⑤ In the Centos C2 container go into the volume and create file

cd /data

touch file3 file4

Come out of the container without Exit (Ctrl+P, Ctrl+Q)

⑥ Create another centos container C3 and this container shou use the volume used by C2

docker run --name C3 -it --volumes-from C2 centos

⑦ In the centos C3 container go into the volume and create files

cd /data

touch file5 file6

Come out of the container without Exit (Ctrl+P, Ctrl+Q)

⑧ Go into any of the 3 containers and we should see all the files

docker attach C1 (or) C2 (or) C3

ls

exit

⑨ Identify the mounted location

docker inspect C1

Go to "Mounts" section and copy the "Source" path.

⑩ Delete all the 3 containers

docker rm -f C1 C2 C3

⑪ Check if the files are still available on the host machine

cd "source path copied from step 9"

ls

Docker volume containers: These are bidirectional i.e; data from the host machine can be copied into the containers and from the container we can save data the on the host machine

Use case: Create a docker volume on host machine and create some files in it.

Mount this volume on an Ubuntu container and check if these files are available on the container

① Create a docker volume

```
docker volume create myvolume
```

② To get detailed info about the volume

```
docker volume inspect myvolume
```

③ Copy the path shown in "Mountpoint" in above output and move into it

```
# cd "Mountpoint_path_copied_in_step2"
```

④ Create some files here

```
touch file1 file2
```

⑤ Mount this volume on an Ubuntu container

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

⑥ In the Ubuntu U1 container go into /tmp folder and check if files from host machine are available

```
# cd /tmp
  ls
```