**Docker**

**Images:** A docker image is a collection of binaries and libraries that are necessary for an application to run.

**Container:** A Running instances of docker is called as Container.

**Docker Host:** The Machine were docker is installed. All the docker container runs on this machine it can be Windows, Linux etc.

**Docker Client:** This is a piece of S/W which is responsible for accepting the docker commands and it passes these commands to a background process.

**Docker daemon:** This is background process which runs on docker engine and it accepts the commands that are come from docker client and execute them on the images or containers or the docker repository.

**Docker Registry:** This is the location where all the docker images are stored this is of two types: 1) Public 2) Private.

**Docker Commands:**

1. To download the image from registry **docker pull image\_name.**
2. To search for a docker image on dockerhub **docker search image\_name.**
3. To see the list of docker images that are available in our docker host **docker images (or) docker images ls**
4. To delete a docker image **docker rmi image\_name**
5. To Create an image from docker file **docker build –t new\_image\_name .**
6. To Create an image from a Container **docker commit container\_name/Container\_id new\_image\_name**
7. To upload the docker images into dockerhub **docker push**

**Working On Containers:**

1. To start the stopped Container **docker start Container\_name/Container\_id**
2. To stop a running Container **docker stop Container\_name/Container\_id**
3. To restart a running Container **docker restart Container\_name/Container\_id**
4. To restart after 10 seconds **docker restart –t 10 Container\_name/Container\_id**
5. To find detailed info about a Container **docker inspect Container\_name/Container\_id**
6. To find the logs generated by a Container **docker logs Container\_name/Container\_id**
7. To see the ports that are working in a container **docker port Container\_name/Container\_id**
8. To delete a stopped Container **docker rm Continer\_name/Container\_id**
9. To delete a running Container **docker rm –f Container\_name/Container\_id**
10. To stop all running Container **docker stop $(docker ps –aq)**
11. To remove all stopped Container **docker rm $(docker ps –aq)**
12. To remove all running Containers **docker rm –f $(docker ps –aq)**
13. To Create a new Container from docker image **docker run image\_name**

**Run Command Options:**

1. **–it** To open interactive terminal in a container
2. **–d** Runs Container in detached mode as a Container
3. **–name** is used to give a name to Container
4. **–e** Used to pass environment variables to Containers
5. **–network** used to run the container on a specific network
6. **–p** Used for port mapping i.e it will link the container port with the docker host machine port (E.g. –p 8080:80 here 80 is container port (internal port) and 8080 is docker host port (external port))
7. **–P** Used for automatically linked with host machine port and this host port will be some number greater than 30000
8. **–V** Used for attaching volumes to a Container
9. **–Volumes-from** used for creating volumes that are reusable
10. **–memory** used for allocating fixed amount of ram to containers
11. **–CPUS**  Used for CPUallocation
12. **–rm** will delete the container on exit
13. **–-link** Used to link 2 Containers
14. To go into a container that is running in background with an interactive terminal **docker attach container\_name/Container\_id**
15. To run a specific process in a container (Other than the default process) **docker exec –it container\_name/Container\_id command to be executed**
16. To see the list of all running containers **docker container ls**
17. To see the list of all Containers (running & Stopped) **docker ps –a**

**Docker Networking:**

1. To see the list of all docker networks **docker network ls**
2. To find the detailed info about a network **docker inspect network\_name/network\_id**
3. To create a network **docker network create network\_name**
4. To delete a network **docker network rm network\_name/network\_id**
5. To attach running containers to a network **docker network connect network\_name/network\_id container\_name/Container\_id**
6. To dittach a container from a network **docker network disconnect network\_name/network\_id container\_name/Container\_id**
7. To delete the all unused containers, volumes and networks and etc **docker system prune –af**

**Docker Volumes:**

1. To see the list of volumes available **docker volume ls**
2. To Create a new volume **docker volume create volumename**
3. To delete a volume  **docker volume rm volumename/volumeid**
4. To find detailed info about a volume **docker volume inspect volumename/volumeid**

**Examples:**

1. **docker run --name webserver –p 8080:80 –d nginx**

**Docker-Compose:**

To create multiple container orchestration. docker-compose simplify this activity by using the yamel file. This yamel file is reusable for starting and stopping the services for this we need to install the docker compose

**Syntax:**

---

Version: ‘3’

Services:

devserver:

image: Jenkins

ports:

* 5050:8080

qaserver:

image: tomcat

port:

* 6060:8080

links:

* devserver

Prodserver:

Image: tomcat

Port:

* 7070:8080

links:

* devserver

...

**Docker File:**

This is another way of creating customized docker images based on certain predefined keywords.

**FROM:** This is used to specify the base image from which the docker file has to be create.

**MAINTAINER:** This is to specify the name of the organization or the author that has created the docker file

**CMD:** Used for firing the command automatically whenever the container starts

**ENTRYPOINT:** This is also used for firing the command whenever a container starts. It can also be used to specify the default process that should be executed when the container starts.

**RUN:**  Used for running linux commands with in the container it is mainly useful in package management in containers i.e., updating the repositories, installing uninstalling the packages.

**VOLUME:** Useful to attach default volumes to a container

**EXPOSE:** This is used for specify the port that should be exposed in the container

**COPY:** Useful copying files from docker host to container

**ADD:** This can also be used for copying files from host to container. It can also be used for downloading the remote urls.

**LABEL:**  Useful in giving label to the image

**STOPSIGNAL:** Use to specify the key strock that will terminate the container

**WORKDIR:**  Useful in defining the default working directory in the container

**Sample Program:**

FROM Centos

MAINTAINER intelliq

CMD [“date”]

CMD [“ls”,”-la”]

ENTRYPOINT [“/bin/bash”]

RUN apt-get update

RUN apt-get install git –y

USER root

To build the image **docker build –t name .**

**Cache Busting**

If we create a build from image the docker will read there cache whether these steps are previously executed or not if it executed previously, it will take data from the cache memory, only for new arguments it will take from docker hub

**Disadvantage:**

1. if your installing by using old image it will take data from cache i.e., while creating image which is latest version of s/w that version it will download irrespective of dockerhub.

Example program to avoid this

FROM ubuntu

MAINTAINER prasanth

RUN apt-get update && apt-get install git maven tree

**DOCKER VOLUMES**

Docker containers are ephemeral (Temporary) data i.e., they are created for specific purpose onces it is done they are deleted but the data in the containers processes should be permanent. This is of two types:

1. Simple Docker Volume
2. Docker volume Container

**Simple Docker Volume:** This is used for preserving the data of the container even after the container is deleted. This data cannot be reused by other containers.

Syntax: docker run –name c1 –it –v /data ubuntu

Eg:

1. Create a folder Mkdir /data (name should be data)
2. Start a container by using above syntax
3. Go data folder in the container and create some file
4. Exit from container to identify the mount location inspect the container
5. Delete the container
6. Cd path of the source path which is taken form 4th step

**Docker Volume Containers:** These are reusable volumes which can be shared b/w multiple containers.

Eg:

1. Create a directory mkdir /data
2. Start the container by using the syntax **docker run –name –it c1 –v /data ubuntu**
3. Go to data folder in container and create some file and exit from the container by using ctrl+p ctrl+q
4. Start another container that should use the first container mount point Syntax **docker run –-name -it c2 –-volumes-from c1 ubuntu**
5. Go to data folder and create some files come out of the container same as step3
6. Same as 4 and 5 for another container
7. Docker attach c1/c2/c3 in any container
8. Docker inspect c1/c2/c3
9. Delete three containers and check in local /data folder.

**Docker Registry**

Docker images can be pushed into docker registry in two ways 1) Public

2) Private

**1) Public:**

1)Create docker registry usename and password

2) create a image from the container

3) Tag that image **docker tag imageid username/imagename:tagname**

4) To push the image **docker push dockerusername/imagename**

**2) Private:**

It can be created by using docker image called registry. Onces the docker registry image is started as a container it will behave as private registry.

1. Start registry as a container **docker run –-name localregistry –d –p 5000:5000 registry**
2. Pull the image **docker pull alpine**
3. Tag the image **docker tag alpine localhost:5000/alpine**
4. Push the image **docker** **push localhost:5000/alpine**

**DOCKER NETWORKING**

Docker networking are 4 types:

1. bridge
2. host only
3. null
4. Overlay

Bridge n/w is the default n/w that is used by containers that are running on same docker host all the containers present in the same bridge network can communicate with each other

Host only is the n/w used when we want to create containers that should be communicate with only host machine and not with other containers

Null N/W is used for creating isolated containers these containers cannot communicate with host machine or with other containers

Overlay n/w is used when the containers are running in distributed environment on multiple docker host and all these containers will communicate with each other

1. Create 2 docker networks
2. Create a container with network 1 and image name **docker run –-name c1 –it –-network intelliq1 busybox**
3. Come back from the container by ctrl+p ctrl+q
4. Repeat the step 2 again with intelliq1 c2,intelliq2 c3
5. Go to c2 and ping c1 ip address
6. To connect c2 with intelliq2 by using **docker network connect intelliq2 c2**
7. Go to container c3 ping c2 ip address it will ping