Docker:

Virtualization: Here we have a bare metal (H/W) on top of which we install the host OS. On the host OS we install an application called hypervisor (VMware, ESXI, Citrix Xen, Hyper-V). On the hypervisor we can install any guest OS and on the guest OS we can install the applications that we want.

The problem here is, these applications must pass through many layers in order to access the h/w resources.

<table>
<tr><td>Oracle APP</td><td>MS SQL</td></tr>
<tr><td>Guest OS</td><td>Guest OS (Windows)</td></tr>
<tr><td colspan="2">Hypervisor</td></tr>
<tr><td colspan="2">Host OS (ubuntu)</td></tr>
<tr><td colspan="2">Bare metal</td></tr>
</table>

Containerization:

Here we have a bare metal on which the Host OS is installed, on the host OS we install an application called Docker engine. On the Docker engine we can run any application. These applications have to pass through less number of layers in order to access the hardware resources

<table>
<tr><td>Oracle APP</td><td>MS SQL app</td></tr>
<tr><td colspan="2">Docker Engine</td></tr>
<tr><td colspan="2">Host OS Ubuntu</td></tr>
<tr><td colspan="2">Bare Metal</td></tr>
</table>

Docker performs *“process isolation”* ie. it removes the dependency that an application on an OS and it allows that application to run directly on the docker engine. Docker can spin up the necessary environment be it dev environment or testing environment or prod environment in a matter of seconds.

Docker can be used at all the stages of Build, Ship and Run. ie. for developing applications, testing and building them and finally running them in prod environment.

Docker comes in 2 flavors

1. CE(*Community Edition)*
2. *EE(Enterprise Edition)*

Docker Images and containers

A docker image is a collection of binaries and libraries which are necessary for one software application to run. A running instance of an image is called as a container. Any number of containers can be created from one image.

Docker Host

The machine on which docker is installed is called as the docker host. It can be Windows, Linux or Mac.

Docker client

This is an application which is part of the docker engine which is responsible for accepting the docker commands from the user and pass it to docker daemon.

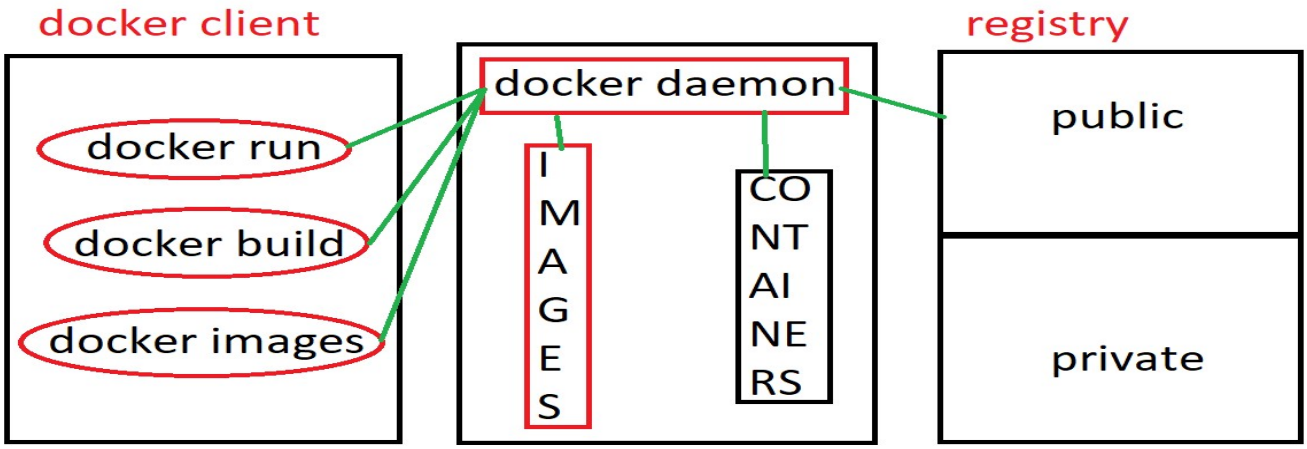
Docker Daemon

This is a background process which is also a part of docker engine and the responsibility of docker daemon is to accept the docker commands from the docker client and depending on the kind of command route them to docker images or containers or the docker registry

Docker registry

This is the location where all the docker images are stored. It is of two types. Public and Private. Public registry is hub.docker.com and images uploaded here can be accessed by anyone. Private registry is

setup within our servers using a docker image called registry and only our organization team members can access the registry.



Working on docker images

To download a docker image

```
docker pull image_name
```

To upload a docker image into the registry

```
docker push image_name
```

To see the list of all docker images available in docker host

```
docker image ls (or) docker images
```

To search for an image in the registry from the command prompt

```
docker search image_name
```

To tag an image with a registry

```
docker tag image_name registry_id/new_image_name
```

To create a new image from a docker container

```
docker commit container_name/container_id new_image_name
```

To create a new image from a dockerfile

```
docker build –t new_image_name .
```

To delete all unused images

```
docker system prune –a
```

To delete a specific unused image

```
docker image rm <imagename>
```

To delete all unused images without atleast one container associated to them

```
docker image prune -a
```

Working on Docker containers

To start a stopped container

```
docker start container_name/container_id
```

To stop a running container

```
docker stop container_name/container_id
```

To remove a stopped container

```
docker rm container_name/container_id
```

To remove a running container

```
docker rm –f container_name/container_id
```

To restart a running container

```
docker restart container_name/container_id
```

To restart after 30 seconds

```
docker restart –t 30 container_name/container_id
```

To stop all running containers

```
docker stop $(docker ps –aq)
```

To see all stopped containers

```
docker ps
```

To delete all stopped containers

```
docker rm $(docker ps –aq)
```

To delete all containers (running as well as stopped)

```
docker rm –f  $(docker ps –aq)
```

To see the logs generated by a container

```
docker logs container_name/container_id
```

To see the ports opened by the container

```
docker port container_name/container_id
```

To come out of the shell of a container without exit

```
Ctrl+p, ctrl+q
```

To reenter into the shell of that container

```
docker attach container_name/container_id
```

To get detailed info about any container

```
docker inspect container_name/container_id
```

To execute any command in a container from the docker host

```
docker exec –it  cont_name/cont_id command to be executed
```

Eg: To open interactive bash shell in a container

```
docker exec –it container_name/container_id bash
```

To see the list of all the running containers

```
docker container ls
```

To see the list of all the containers (running as well as stopped)

```
docker ps –a
```

To create a new container

```
docker run image_name
```

Run command options

--name	Used to give a name for the container
-it	Used for opening interactive terminal in the container
-d	Used to run the container in detached mode as a background process
-v	Used for attaching an external directory or device as a volume
--volumes-from	Used to create sharable volumes which can be used by multiple containers
-p	Used for port mapping. It will help the container port (internal port) with a port on the docker host (external port) Eg: -p 8080:80 Here 8080 is a container port and 80 is the host port
-P	Used for automatic port mapping ie the internal port of the container will be linked with some port number which is greater than 30000
--link	Used for linking multiple containers for creating the micro services architecture
-e	Used for passing environment variables to the container
-rm	This will delete the container on exit
--network	This is used to specify on which network these containers should run
--memory	Used for a fixed amount of memory allocation to the containers
--cpus	Used for specifying how many cpu's should be used by the container

Working on docker networking

To see the list of networks

```
docker network ls
```

To get detailed info about a network

```
docker network inspect network_id/network_name
```

To create a network

```
docker network create --driver network_type network_name
```

To attach a running container to a network

```
docker network connect net_id/net_name cont_name/cont_id
```

To remove a container from a network

```
docker network disconnect net_id/net_name cont_id/cont_name
```

To delete a network

```
docker network rm network_id/network_name
```

Working on Docker volumes:

To see the list of all the docker volumes

```
docker volume ls
```

To create a new docker volume

```
docker volume create volume_name
```

To get detailed info about a volume

```
docker volume inspect volume_name/volume_id
```

To delete a volume

```
docker volume rm volume_name/volume_id
```

To delete all unused volumes

```
docker system prune –volumes
```