

## **DOCKER**

- Docker is an opensource centralized platform designed to create, deploy and run applications.
- Docker uses container on the host OS to run applications. It allows applications to use same Linux kernel as a system on the host computer rather than creating a whole virtual OS.
- We can install docker on any OS but docker engine runs natively on Linux distributions.
- Docker written in “GO” programming language.
- Docker is a tool that performs OS level virtualization also known as Containerization.
- Before docker many users face the problem that a particular code is running in the developer’s system but not in the user’s system.
- Docker was first release in march 2013. It is developed by Solomon Hykes and Sebastian Pahl.
- Docker is a set of Platform-as-a-Service that uses OS level virtualization whereas VMWare uses hardware level of virtualization.

### **Advantages of Docker:**

- No pre allocation of RAM.
- CI efficiency: - docker enables you to build a container image and use that same image across every step of the deployment process.
- Less cost.
- It is light in weight.
- It can run on physical h/w / virtual h/w or on cloud.
- You can reuse this image.
- It took very less time to create container.

### **Disadvantages of Docker:**

- Docker is not a good solution for application that requires rich GUI.
- Difficult to manage large number of containers.
- Docker doesn’t provide cross platform compatibility means if an application is designed to run in a docker container in windows than it can’t run in Linux or vice-versa.
- Docker is suitable when the development O.S and testing O.S are same. If the O.S are different then we should use VM.
- No solution for data recovery and backup.

### **Components of Docker:**

#### **A. Docker Daemon:**

- Docker daemon runs on host O.S.
- It is responsible for running containers to manages docker services.
- Docker daemon can communicate with other daemons.

B. Docker Client:

- Docker users can interact with docker through a client.
- Docker client uses commands and REST API to communicate with the docker daemon.
- When a client runs any server command on the docker client terminal, the client terminal sends these docker commands to the docker daemon.
- It is possible for docker client to communicate with more than one daemon.

C. Docker Host:

- Docker host is used to provide an environment to execute and run applications.
- It contains the docker daemon, images, containers, networks and storages.

D. Docker Hub/ Registry:

- Docker registry manages and stores the docker image.
- There are two types of registries in the docker:
  - a. Public Registry: it is also called as docker hub.
  - b. Private Registry: it is used to share image with in the enterprise.

E. Docker Image:

- Docker images are the read only binary templates used to create docker containers.
- or
- Single file with all the dependencies and configuration required to run a program.

Ways to Create an Image:

- a. Take image from the docker hub.
- b. Create image from docker file.
- c. Create image from existing docker containers.

F. Docker Containers:

- Containers hold the entire packages that is needed to run the application.
- Or
- In other words, we can say that the image is a template and the container is a copy if that template.
  - Container is like a virtual machine.
  - Images becomes container when they run on docker engine.

### **Basic Docker Commands:**

To see all images present in your local repo:

```
# docker images
```

To find out images in docker hub

```
# docker search image_name
```

To download image from dockerhub to local machine

```
# docker pull image_name
```

To give a name to container

```
# docker run -it --name new_name image_name /bin/bash
```

To check service start or not (status)

```
# docker service status
```

To start: #docker service start

To stop: # docker service stop

To start container

```
#docker start container_name
```

To go inside container

```
# docker attach container_name
```

To see all containers

```
# docker ps -a
```

To see running containers

```
# docker ps
```

To stop container

```
# docker stop container_name
```

To delete a container

```
# docker rm container_name
```

Create container from our own Image:

Login into AWS account and start your EC2 instance, access it from putty.

Now we have to create container from our own image. Therefore, create one container first:

```
#docker run -it --name container_name image_name /bin/bash
```

```
#cd tmp/
```

Now create one file inside this tmp directory

```
# touch myfile
```

Now if you want to see the difference between the basic image and the changes on it

```
# docker diff container_name image_name
```

Now create image of this container

```
# docker commit newcontainer_name image_name
```

```
#docker images
```

Now create container from this image

```
# docker run -it --name newcontanier_name image_name /bin/bash
```

```
# ls
```

```
# cd tmp
```

```
# ls (you will get all of your files)
```

### **Dockerfile:**

Dockerfile is basically a text file. It contains some set of instructions. Automation of docker image creation.

Dockerfile components:

FROM: for base image, this command must be on the top of the dockerfile.

RUN: to execute commands, it will create a layer in image

MAINTAINER: author/ owner/ description

COPY: copy files from local system (docker vm) we need to provide source, destination (we can't download file from internet and any remote repo.)

ADD: similar to copy but it provides a feature to download files from internet, also extract file at docker image side.

EXPOSE: to expose ports such as port 8080 for tomcat , port 80 for nginx etc.

CMD: execute commands but during container creation.

ENTRYPOINT: similar to CMD but has higher priority over CMD, first commands will be executed by ENTRYPOINT only.

ENV: environment variables

### **Dockerfile**

- Create a file named Dockerfile
- Add instructions in Dockerfile
- Build dockerfile to create image
- Run image to create container

```
# vi Dockerfile
```

```
FROM ubuntu
```

```
RUN echo "Nagarjuna hota" > /tmp/testfile
```

To create image out of Dockerfile

```
# docker build -t myfile
```

```
#docker ps -a
```

# docker image

Now create container from the above image

#docker run -it --name mycon mying /bin/bash

#cat /tmp/testfile

#vi dockerfile

FROM ubuntu

WORKDIR /tmp

RUN echo "thank you" > /tmp/testfile

ENV myname naga

COPY testfile1 /tmp

ADD test.tar.gz /tmp

### **Docker Volume:**

- Volume is simply a directory inside our container.
- Finally, we have to declare this directory as a volume and then share volume.
- Even if we stop the container still, we can access volume.
- Volume will be created in one container.
- You can declare a directory as a volume only while creating container.
- You can't create volume from existing container.
- You can share one volume across any number of containers.
- Volume will not be included when you update an image.
- You can map volume in two ways:
  - a. Container to container
  - b. Host to container

### **Benefits of Volume:**

- Decoupling container from storage.
- Share volume among different containers.
- Attach volume to containers.
- On deleting container volume doesn't delete.

### **Creating Volume from Dockerfile:**

Create a Dockerfile and write

```
FROM ubuntu
```

```
VOLUME "myvolume"
```

Then create image from this Dockerfile

```
#docker build -t myimage
```

Now create a container from this image and run

```
# docker run -it --name container1 myimage /bin/bash
```

Now do ls, you can see myvolume.

### **Now share volume with another container**

Container to container

```
# docker run -it --name container2 (new) --privileged=true -volumesfrom container1 ubuntu /bin/bash
```

Now after creating container2, myvolume is visible. Whatever you do in one volume, can see from other volume.

```
#touch /myvolume/samplefile
```

```
#docker start container1
```

```
# docker attach container1
```

```
#ls/myvolume
```

You can see sample file here then exit.

### **Now create volume by using command:**

```
#docker run -it --name container3 -v /volume2 ubuntu /bin/bash
```

```
# ls
```

```
#cd /volume2
```

Now create one file cont3file and exit

Now create one more container and share volume2

```
#docker run -it --name container4 --privileged=true --volumefrom container3 ubuntu /bin/bash
```

Now you re inside container do ls you can see volume2

Now create one file inside this volume and then check in container3 you can see that file.

### Volumes (Host to Container)

Verify files in /home/ec2-user

```
#docker run -it --name hostcontainer -v /home/ec2-user:/container --privileged=true ubuntu /bin/bash
```

```
#cd /container
```

Do ls, now you can see all files of host machine.

```
#touch contanerfile (in container) and exit
```

Now check in EC2 machine you can see this above file.

### Some other commands:

```
#docker volume ls
```

```
#docker volume create <volumename>
```

```
#docker volume rm <volumename>
```

```
#docker volume prune (it removes all unused docker volume)
```

```
#docker volume inspect <volumename>
```

```
#docker container inspect <containername>
```

### Docker Port Expose:

Login into AWS account, create one linux instance.

Now go to putty -> login as -> ec2-user

```
#sudo su
```

```
# yum update -y
```

```
# yum install docker -y
```



```
# service docker start
# docker run -td --name techserver -p 80:80 ubuntu
# docker ps
# docker port techserver o/p- 80/tcp - 0.0.0.0/80
# docker exec -it techserver /bin/bash
# apt-get update
# apt-get install apache2 -y
# cd /var/www/html
# echo "write some msg" > index.html
#service apache2 start
# docker run -td --name myjenkins -p 8080:8080 jenkins
```

Difference between docker attach and docker exec:

- Docker 'exec' creates a new process in the container's environment while docker 'attach' just connect the standard input/output of the main process inside the container to corresponding standard input/output error of current terminal.
- Docker 'exec' is specifically for running new things in an already started container be it a shell or some other process.

What is the difference between docker expose and publish:

Basically you have three options:

1. Neither specify expose nor -p
  2. Only specify expose
  3. Specify expose and -p
- 
1. If you specify neither expose nor -p, the service in the container will only be accessible from inside the container itself.
  2. If you expose a port, the service in the container is not accessible from outside docker but from inside other docker containers so this is good for inter-container communication.
  3. If you expose and -p a port, the service in the container is accessible from anywhere even outside docker.

If you do -p but do not expose docker does an implicit expose. This is because if a port is open to the public, it is automatically also open to the other docker containers. Hence -p includes expose.

### **How to push docker image in docker hub:**

Go to AWS account – select Amazon linux

Now go to putty – login as – ec2-user

```
#sudo su
```

```
#yum update -y
```

```
#yum install docker -y
```

```
#service docker start
```

```
#docker run -it ubuntu /bin/bash
```

Now create some files inside container, now create image of this container

```
#docker commit container1 image1
```

Now create account in hub.docker.com

Now go to EC2 instance

```
#docker login
```

Enter your username and password

Now give tag to your image

```
#docker tag image1 dockerid/newimage
```

```
#docker push dockerid/newimage
```

Now you can see this image in docker hub account

Now create one instance in another region and pull image from hub

```
#docker pull dockerid/newimage
```

```
#docker run -it --name mycon dockerid/newimage /bin/bash
```

Some important commands:

Stop all running containers: # docker stop \$(docker ps -a -q)

Delete all stopped containers: # docker rm \$(docker ps -a -q)

Delete all images: docker rmi -f \$(docker images -q)