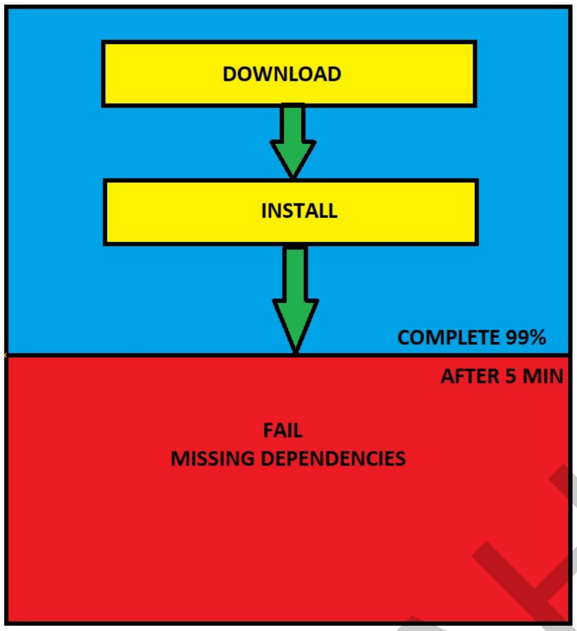
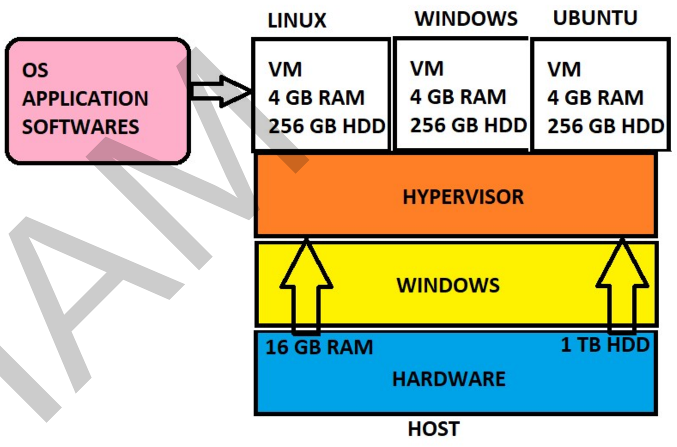
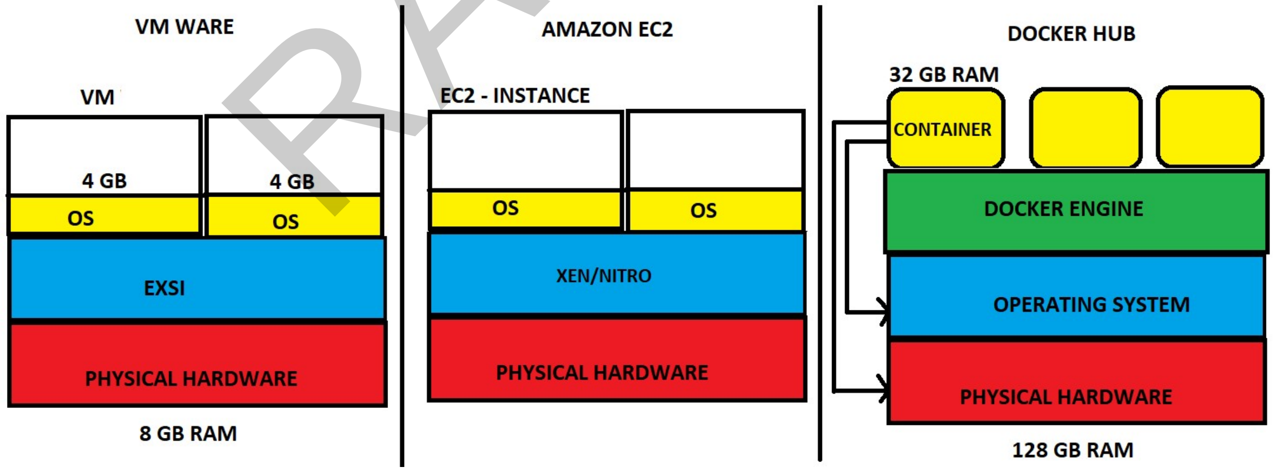
DOCKER

CONTAINER	: It is like a Virtual Machine and It does not have any OS.
VIRTUVALIZATION	: process that allows for more efficient utilization of physical computer hardware and is the foundation of cloud computing.
HYPERVISOR	: Help to make virtualization and to create a VM
CONTAINARIZATION	: Process of packing application along with its dependencies
DOCKER	: It is a tool that create this Container. It is advance than Virtualization.



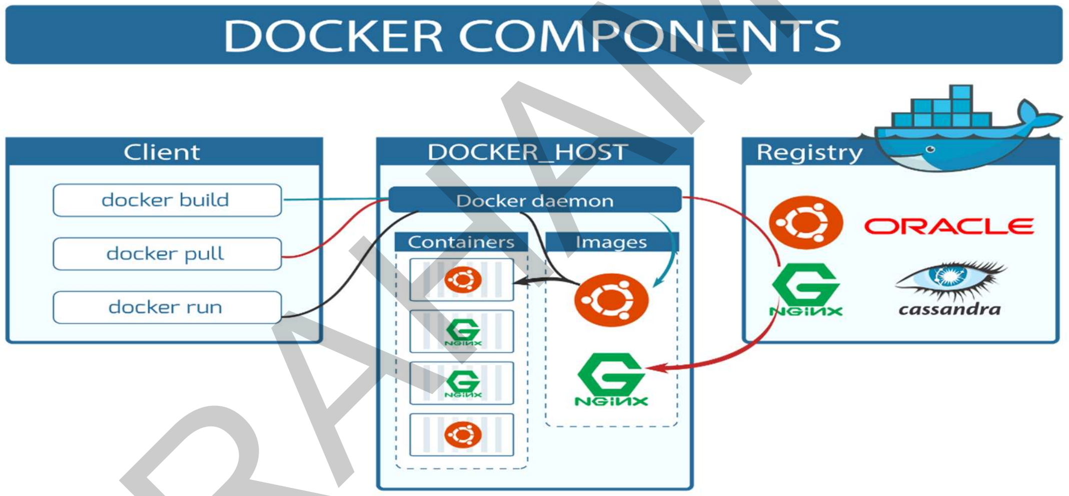




O.S LEVEL VIRTUALIZATION

- It is an opensource centralized platform designed to create, deploy and run applications.
- Docker is written on Go language.
- Docker uses container on host O.S to run applications. It allows applications to use same Linux kernel as a system on the host computer, rather than creating a whole virtual O.S.
- We can install Docker on any O.S but docker engine runs natively on Linux distribution.
- Docker performs O.S level Virtualization also known as Containerization.
- Before Docker many user face problems that a particular code is running in the developer's system but not in the user system.
- It was initially release in March 2013, and developed by Solomon hykes and Sebastian pahl.
- Docker is a set of platform-as-a-service that use O.S level Virtualization, where as VM ware uses Hardware level Virtualization.
- Container consists O.S files but its negligible in size compared to original files of that O.S.

ARCHITECTURE



- **DOCKER CLIENT**: is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to docker daemon, which carries them out. The docker command uses the Docker API.

- **DOCKER HOST**: Docker host is the machine where you installed the docker engine

- **DOCKER DEAMON**: Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage.

- **DOCKER REGISTRY**: A Docker registry is a scalable open-source storage and distribution system for docker images.

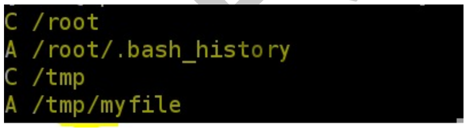
BASIC DOCKER COMMANDS

To install docker in Linux	: yum install docker -y
To see the docker version	: docker - -version
To start the docker service	: service docker start
To check service is start or not	: service docker status
To check the docker information	: docker info
To see all images in local machine	: docker images
To find images in docker hub	: docker search image name
To download image from docker hub to local	: docker pull image name
To download and run image at a time	: docker run -it image name /bin/bash
To give names of a container	: docker run -it --name raham img-name /bin/bash
To start container	: docker start container name
To go inside the container	: docker attach container name
To see all the details inside container	: cat /etc/os-release
To get outside of the container	: exit
To see all containers	: docker ps -a
To see only running containers	: docker ps (ps: process status)
To stop the container	: docker stop container name
To delete container	: docker rm container name

you can create multiple containers by using single image
start vs stop and kill vs remove

TO BUILD IMAGE FROM CONTAINER

- Create a container from our own image.
- Therefore, create a container first.
- docker run -it - -name raham ubuntu /bin/bash
- Cd tmp/ and create a file inside it by using touch myfile and exit from container.
- Now if you want to see the difference between base image and changes on it then,
- docker diff raham update-image (A: append, C: changed, D: deletion)



```
C /root
A /root/.bash_history
C /tmp
A /tmp/myfile
```

- Now create image of this container by using
  docker commit container-name update-image.
- Now list images by using docker images
- Now create container for the image using
  docker run -it - -name remo update-image /bin/bash
- Now go to tmp folder and give ls you will see myfile there.

DOCKER FILE

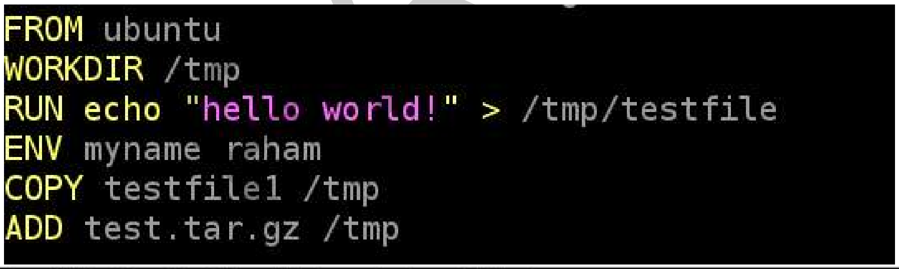
- It is basically a text file which contains some set of instructions.
- Automation of Docker image creation.
- Always D is capital letter on Docker file.
- And Start Components also be Capital letter.

DOCKER FILE COMPONENTS

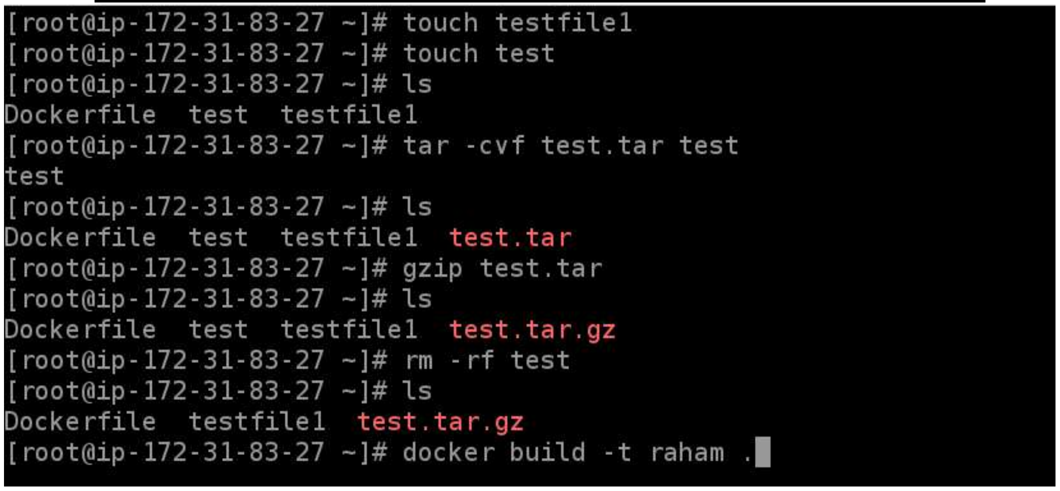
FROM: For base image this command must be on top of the file.
RUN: To execute commands, it will create a layer in file.
COPY: Copy files from local system (docker VM) where need to provide Source and Destination.
ADD: It can download files from internet and also, we can extract file at docker image side.
EXPOSE: To expose ports such as 8080 for tomcat and port 80 nginx etc.
WORKDIR: To set working directory for the Container.
CMD: Executes commands but during Container creation.
ENTRYPOINT: High priority than CMD, because first command will be executed by Entry point only.
ENV: Environment Variables.
ARG: Defines a variable that can be used to build a Docker image
Difference b/w Copy and Add is in copy we can't download file from internet and any remote repo.

FILE CREATION

- Create a file called Docker file and Add instructions in Docker file.
- Build Docker file to create image.
- Run image to Create Container.
  Vi Docker file
  FROM ubuntu
  RUN   echo "Hello world!" > /tmp/testfile
- To create image out of Docker file
  docker build -t image-name . (. = current directory)
  docker ps -a and docker images.
- Now create container from the above image.
  docker run -it - -name container-name image-name /bin/bash
- Cat /tmp/testfile

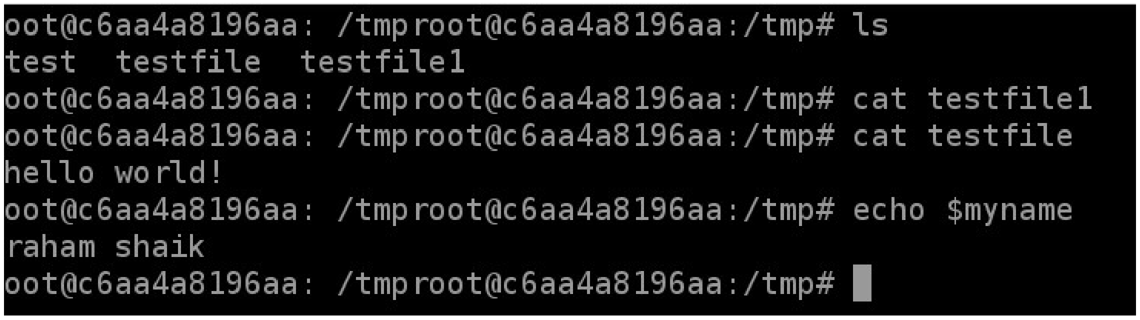


```
FROM ubuntu
WORKDIR /tmp
RUN echo "hello world!" > /tmp/testfile
ENV myname raham
COPY testfile1 /tmp
ADD test.tar.gz /tmp
```



```
[root@ip-172-31-83-27 ~]# touch testfile1
[root@ip-172-31-83-27 ~]# touch test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1
[root@ip-172-31-83-27 ~]# tar -cvf test.tar test
test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1  test.tar
[root@ip-172-31-83-27 ~]# gzip test.tar
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1  test.tar.gz
[root@ip-172-31-83-27 ~]# rm -rf test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  testfile1  test.tar.gz
[root@ip-172-31-83-27 ~]# docker build -t raham .
```

Give docker run -it - -name container name image-name /bin/bash



```
oot@c6aa4a8196aa: /tmproot@c6aa4a8196aa:/tmp# ls
test  testfile  testfile1
oot@c6aa4a8196aa: /tmproot@c6aa4a8196aa:/tmp# cat testfile1
oot@c6aa4a8196aa: /tmproot@c6aa4a8196aa:/tmp# cat testfile
hello world!
oot@c6aa4a8196aa: /tmproot@c6aa4a8196aa:/tmp# echo $myname
raham shaik
oot@c6aa4a8196aa: /tmproot@c6aa4a8196aa:/tmp#
```

DOCKER VOLUMES

- When we create a Container then Volume will be created.
- Volume is simply a directory inside our container.
- First, we have to declare the directory Volume and then share Volume.
- Even if we stop the container still, we can access the volume.
- Volume will be created in one Container.
- You can declare directory as a volume only while creating container.
- We can't create volume from existing container.
- You can share one volume across many number of Containers.
- Volume will not be included when you update an image.
- If Container-1 volume is shared to Container-2 the changes made by Container-2 will be also available in the Container-1.
- You can map Volume in two ways
    1. Container < ------ > Container
    2. Host   < ------- > Container

USES OF VOLUMES

- Decoupling Container from storage.
- Share Volume among different Containers.
- Attach Volume to Containers.
- On deleting Container Volume will not be deleted.

CREATING VOLUME FORM DOCKER FILE

- Create a Docker file and write
  FROM ubuntu
  VOLUME ["/myvolume1"]
- Docker build -t my-image . (Current directory)
- Docker run -it - -name container1 myimage /bin/bash
- Now do ls and you will see myvolume-1 add some files there
- Now share volume with another Container
  docker run -it - -name container2(new) - -privileged=true - -volumes-from container1 ubuntu
- Now after creating container2, my volume1 is visible.
- Whatever you do in volume1 in container1 can see in another container.
- touch /myvolume1/samplefile1 and exit from container2.
- docker start container1
- docker attach container1
- ls/volume1 and you will see your samplefile1.

CREATING VOLUME BY USING COMMAND

- docker run -it - -name container3 -v /volume2 ubuntu /bin/bash
- now do ls and cd volume2
- Now create one file and exit.
- Now create one more container, and share Volume2
  Docker run-it - -name container4 - - -privileged=true - -volumes-from container3 ubuntu
- Now you are inside container and do ls, you can see the Volume2
- Now create one file inside this volume and check in container3, you can see that file.

VOLUMES (HOST - CONTAINER)

- Verify files in /home/ec2-use
- Docker run -it - -name hostcont -v /home/ec2-user:/raham - -privileged=true ubuntu
- Cd raham [raham is (container-name)]
- Do ls now you can see all files of host machine.
- Touch file1 and exit. Check in ec2-machine you can see that file.

SOME OTHER COMMANDS

- docker volume ls
- docker volume create <volume-name>
- docker volume rm <volume-name>
- docker volume prune (it will remove all unused docker volumes).
- docker volume inspect <volume-name>
- docker container inspect <container-name>

DOCKER PUSH

- Select an image which include docker and S.G SSH and HTTP enable anywhere on it.
- docker run -it ubuntu /bin/bash
- Create some filed inside container and create image from that container by using
  docker commit container-name image1
- now create docker hub account
- Go to Ec2 and login by using docker login.
- Enter username and password.
- Now give tag to your image, without tagging we can't push our image to docker.
  docker tag image1 rahamshaik/new-image-name (ex: project1)
- docker push rahamshaik/project1
- Now you can see this image in docker hub account.
- Now create one instance in another region and pull image from hub.
  docker pull rahamshaik/project1
  docker run -it - -name mycontainer rahamshaik/project1 /bin/bash
- Now give ls and cd tmp and ls you can see the files you created.
- Now go to docker hub and select your image -- > settings -- > make it private.
- Now run docker pull rahamshaik/project1

- If it denied then login again and run it.
- If you want to delete image settings -- > project1 -- > delete.

SOME IMPORTANT COMMANDS

To stop all the containers	: docker stop $(docker ps -a -q)
To delete all the stopped containers	: docker rm $(docker ps -a -q)
To delete all images	: docker rmi -f $(docker images -q)