Docker

What is Hypervisor?

Hypervisor is a software that helps to do the virtualization, It shares the physical resources like RAM, CPU, Storage with VMs.

All the VMs on same physical server are isolated because of the individual OS that those have.

The problem of wasting the resources is solved using VM to some extent, Still the VMs are not utilized fully, To reduce the wastage of resources containers are used. Containers have it’s own drawbacks.

VMs are more secured compared to containers because VMs has complete OS, those are isolated from each other. Whereas Containers doesn’t have complete OS, one or other way these containers talk to each other, Shares some resources from Host OS.

Containerization can be done on top of Physical servers (Model 1) or on top or Virtual Machines (Model 2). Model2 is mostly used now a days.

What is Container?

Container is nothing but a package of Application, libraries (application dependencies), system dependencies etc.

Containers are light weight. Because it doesn’t have full operating system. It has the minimalistic system dependencies and share the libraries from Host OS.

What are the main difference between container and VM?

1. **Resource utilization:** Containers share host operating system kernel, makes them light weight faster. VMs have full fledge operating system makes them resource-intensive.
2. **Portability:** Containers are designed to be more portable and can run on any system with compatible host os. VMs are less portable as they need a compatible hypervisor to run.
3. **Security:** VMs has high level of security as each VM has it’s own operating system and can be isolated from the host and other VMs. Containers are less secured as they share host operating system.
4. **Management:** Managing containers is easier than managing VMs as containers are light weight and fast-moving.

**Docker Life Cycle:**

Docker files consists of the docker commands, Docker image can be created by building docker commands, Once the Image is available, We can create containers.

Dockerfile 🡪 Docker Image 🡪 Docker Container

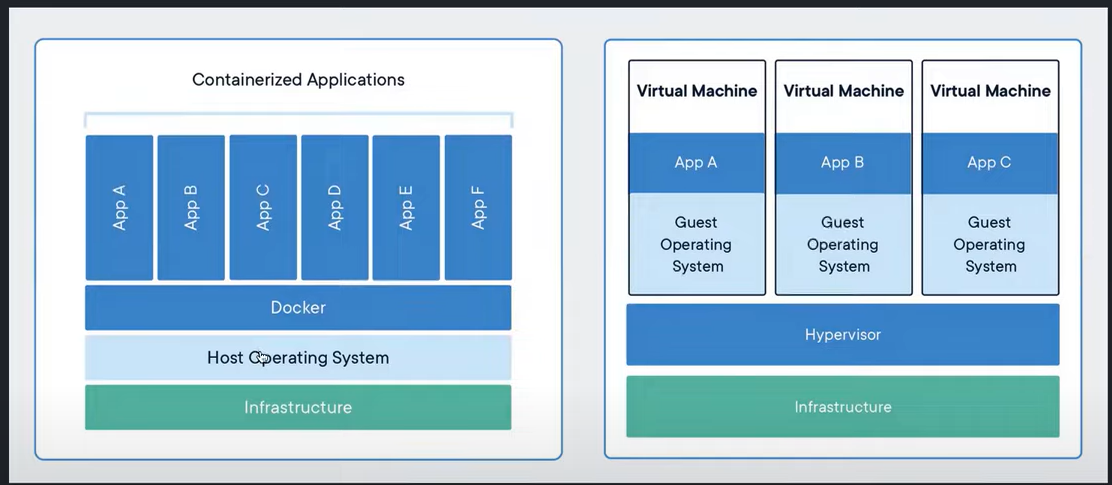
To build image from Dockerfile

docker build -t image\_name:tag path\_to\_dockerfile\_directory

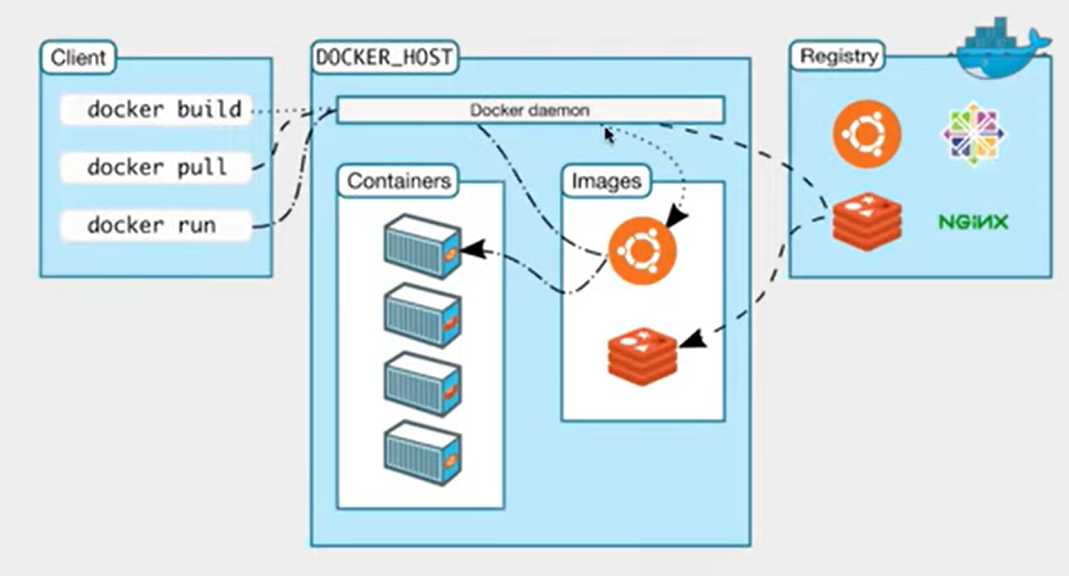
To create docker container from docker image

docker run -d –-name my\_container my\_image:tag

Docker engine is single point of failure. If it goes down, we will not be able to communicate with containers.



Docker is containerization platform to containerize the application.



Docker daemon is heart of the docker. It receives the commands from client and performs the operations. If docker daemons goes down, Containers doesn’t work.

Once the docker image is created, Anyone in the world can download/pull the docker image and creates container and run the application without any dependencies installed explicitly.

**Docker registry:**

It is the place the docker images can be shared with public or private. Dockerhub is one of the docker registery.

To install docker in Virtual Machine(Ubuntu):

Log into the Virtual Machine instance. Run following commands.

sudo apt update

sudo apt install docker.io -y # -y means yes

To verify whether docker is installed:

sudo systemctl status docker

To provide the permissions to docker daemon as root user

sudo usermod -aG docker ubuntu

Logout and login back to the instance

Now, Run below docker command to pull sample hello-world docker container

docker run hello-world

docker build -t image\_name:tag . #(Dockerfile in the same directory)

#-t means tag

docker run -it image\_name:tag

#-it : interactive terminal

docker push image\_name:tag

docker pull image\_name:tag

Docker Commands:

**FROM:**

From command is used to get the base image from docker hub.

It is the base image on which the docker container need to be built.

Ex: FROM image\_name: tag

If you haven’t mentioned the tag, the latest image will be fetched

**WORKDIR:**

This command will set up the work directory location. Where all the application and dependencies can be copied from host

It works as working directory for the subsequent instructions that run in Dockerfile like RUN, CMD, ENTRY POINT etc.

Ex: WORKDIR /path/devops/app

COPY:

COPY source destination

This command will copy the files from host directory to Working directory

Ex: COPY requirements.txt /app

COPY . . # Copies the all the files from host to /app directory in container.

**RUN:**

This command runs the commands inside the container working directory, To install the required softwares and dependencies for the application to run

Ex: RUN apt-get apt update && \

apt-get install -y python3 python3-pip && \

pip3 install -r requirements.txt

**ENTRYPOINT:**

The commands given in the entrypoint cannot be overridden by user.

Entry point commands will run when the docker run command is executed. i.e when a container based on the image is started.

It defines the default executable for the container.

Ex: ENTRYPOINT [“python3”]

**CMD:**

It can define the default commands that runs when the container based on the images is started.

It can have the executable and commands.

The CMD commands can be overridden at run time by specifying a new command when running the container.

Ex: CMD [“python”, ”app.py”]

How to override the CMD commands written in Dockerfile while run time.

docker run my\_image python another\_script.py

To expose the container port to host run the following command.

docker run -it -p 8000:8000 my\_image

**Multi stage Docker build:**

In first stage we fetch Rich image (Ex: ubuntu) which has all the commands to create a build. After executing all the commands. Build can be created in stage 1.

Whatever the base image we get in stage 1 that doesn’t goes into the docker build. We no need to bother about the size of the base image.

With this approach the size of the docker image can be reduce upto 80%.

**Distroless images:**

It is the minimalistic docker image that only contains runtime.

Ex: python distroless images, golang etc.

Distroless images has only runtime, It provides highest level of security to containers. It prevents from the exposing the vulnerabilities.

FROM scratch

scratch is the minimalistic distroless image that we have till date.

Docker Problems:

Docker containers are ephemeral i.e short lived. When the container goes down the CPU, Memory, Storage that container uses on host OS will be freed up. The log file in the container will be deleted.

If docker doesn’t have persistent storage. It can’t store the data of the container. If container goes down the data is lost.

**Bind mounts:**

It’s binding a specific directory on the host to the specific directory on the container.

If /app is bind between container and Host. They both can access the same directory. Even if container goes down. Host can access it, so that the container information or data is still available. When the container is come up it can access the data.

**Docker Volumes:**

Docker volumes does the same job as bind mounts.

It’s logical partition that is created on the Host.

Volumes has lifecycle like create, destroy volumes.

bind mounts are allowed to bind directory of container with One host directory only whereas Volumes of a containers can be created in the same host or external host/S3/NFS etc.

Volumes can be shared easily from one container to another container.

Volume can be assigned to one or more containers.

Volumes are high performance. For example your host is normal. If you want high I/O operation you can mount a volume with high performance.

docker -v <commands>

dokcer - - mount <commands>

Dokcer - - mount command is uses verbose, It is more clear to user to understand which directory is mounted on host with container.

docker volume create <vol\_name> # To create docker Volume

docker volume ps # To get the docker volume list

docker volume inspect <vol\_name> # gives you the details about volume

docker volume rm <v1> <v2> # To delete the volumes

docker run -d - - mount source=’host\_dir’, target=’container\_dir’ <image\_name>

It mounts the volume to the container and runs the container.

To check the details of the volume, you can run the command

docker inspect <container\_id>/name

search for Mounts, You will find

Source : <> #directory on the Host

Desitination: <> # directory on the container

Volume name

Docker volume is not allowed to delete when it is attached to any container. To delete it first we need to stop the container and delete the volume.

When docker run command is run with - - rm then the volume,container gets deleted automatically when the container is stopped.

Docker Networking:

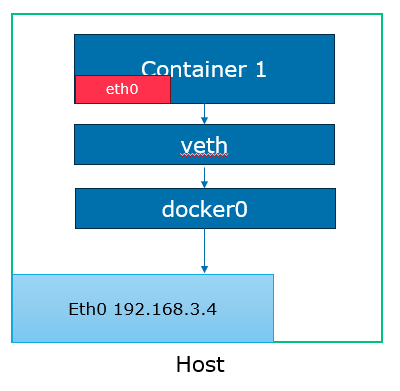
Networking allows containers to talk with each other and host.

By default each container talks to host.

**Usecase 1:** Container 1 communicates to Container 2

Any container, VM by default have the network eth0 : 192.168.3.4 for example.

By default, Bridge networking is available between the container and host. It create docker0 virtual ethernet to establish network between container and host.



You can get the IP address of the container using docker inspect <container\_name>

Use docker exec -it <container\_name> to enter into container. Ping the other container IP from this container.

docker network ls # to get the networks list

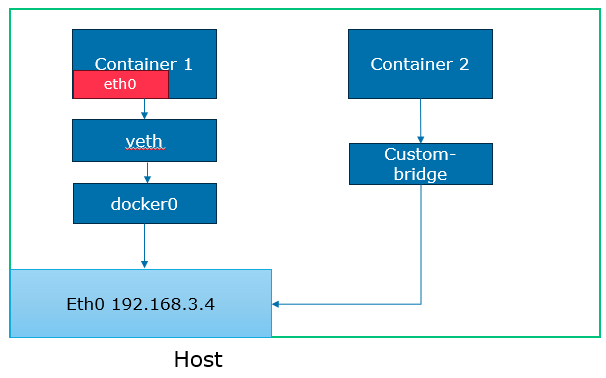
docker network rm <network\_name/id> # to delete network

docker network create secure-network # to create network

To create a container with custom network

docker run -d –name <cont\_name> - - network <network\_name/id> image

**Usecase2:** Container 1 is isolated from container 2



Containers can be isolated using custom bridge networks. We need to provide the network to the container while creating it.

Host Networking:

In this container uses the Host networking to communicate with host.

When the host network is attached to the container. It doesn’t provide any IP to the container. It directly binds the host-network to the container.

Overlay Networking:

It is used to created network in Kubernetes, Docker swarm. When the containers are on multiple host to create a network among the containers in the cluster overlay networking is used.