# DOCKER

Docker is one of the tools that used the idea of the isolated resources to create a container that allows applications to be packaged with all the dependencies installed and ran wherever we wanted.

Docker can only run-on Linux machines this means I can’t install Docker directly on Windows or any other OS.

If I want install Docker on windows then I need to run a Linux VM in windows on top that I need to run Docker.

## Container

A container is a set of isolated processes and resources. Linux achieves this by using namespaces, which allows processes to access only resources in that particular namespace, which allows having a process tree means set of processes that is completely independent of the rest of the systems processes

## Virtualization (VM)

- VM is way of running virtual OS on top a host OS using a special software called **Hypervisor**.

- VM directly shares the hardware of the host OS.

## Diff B/w Virtualization and Containerisation

VM vs Containerisation

1. Virtualization at hardware level 1. Virtualization at OS level

2. Heavyweight - consume more host 2. Lightweight resources

3. VM uses hypervisor 3. containerisation tool is used

4. limited performance - Boot up time      4. Native performance - usually boot

is more which being in minutes   fast in seconds.

5. Consumes more storage 5. Shares OS storage means only uses

required storage.

6. Supports all OS 6. Supports on Linux

## Host Machine

This is the machine in which docker is running

## Docker images

Docker Image is a read only template consists of set instructions to create a container that can run on docker platform

List images in a machine

docker images

To pull / download docker image

docker pull <image\_name>:<tag\_name>

To delete docker image

docker rmi <image\_name>:<tag\_name>

(OR)

docker rmi <image\_id>

To delete all the images

docker rmi $(docker images -q)

To tag a docker image

docker tag <old\_image> <new\_image>

ex: docker tag ubuntu:latest my\_ubuntu:1.0

To delete dangling images / to delete all unwanted images / images which are not used

docker image prune

## Docker containers

A container is a set of isolated resources. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

To create/run a container from image

docker run -it -d --name <container\_name> <image\_name>

-it - Interactive Terminal (tty)

-d - detached mode (whenever we create a container it will auto login to avoid this

we can create a container in detached mode

--name used to provide user defined container name

To list the running containers

docker ps

(OR)

docker container ls

To delete a stop container

docker rm <container\_name>

(OR)

docker rm <container\_id>

To delete a running container

Forceful deletion

docker rm -f <container\_id>

Graceful deletion

docker rm $(docker stop <container>)

To list all the containers (running and stopped)

docker ps -a

To list all stopped containers

docker ps -a --filter status=exited

To delete all stopped container

docker rm $(docker ps -aq --filter status=exited)

To check the logs of containers

docker logs <container\_id>

To run a command inside a container

docker exec -it <container\_id> <command>

To login / get inside a containre

docker attach <container\_id>

## Custom Docker Image / Docker file

- Dockerfile is used to create custom image on top stock image or any other image as base image.

### FROM

- FROM must be the first non-command instruction in the Dockerfile.

- FROM is used to specify base image on top of this image all the next instructions will be executed.

FROM <image\_name>:<tag>

### RUN

The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile

### COPY and ADD

- Both **copy and add** instruction is used to copy files and directories from host machine to the image.

- The source path to copy files should always be evaluated with reference to Dockerfile.

**ADD** supports extra source formats

- If the source is a compressed file add will automatically uncompressed it to the destination.

- If the source is a link to a downloadable file, it will download to the destination.

### CMD and ENTRYPOINT

#### CMD :-

The main purpose of the CMD command is **to launch the software required in a container**. For example, the user may need to run an executable .exe file or a Bash terminal as soon as the container starts – t​he CMD command can be used to handle such requests.

#### Entry Point :-

An ENTRYPOINT instruction is used **to set executables that will always run when the container is initiated**.

- If we use multiple CMD or ENTRYPOINT in the same Docker file only the latest one will be considered and all the other CMD or ENTRYPOINT will be ignored.

- If we use both CMD and ENTRYPOINT in the same Docker file, Entrypoint will get the highest priority and the command of CMD will become as arguments to Entry point

### Difference b/w CMD & Entrypoint

  -CMD command can be overridden at the runtime.

  - ENTRYPOINT can't be overridden at the runtime but the runtime command

    will become parameters to ENTRYPOINT command.

Note: Q. Can we override ENTRYPOINT

Yes, after docker 1.6 version docker has given option to over

  Entrypoint command at the runtime using --entrypoint

### ENV

- This instruction is used to set the environment variable inside the container.

ENV <variable\_name> <value>

ENV <variable\_name>=<value>

multiple

ENV<variable\_name>=<value><variable\_name>=<value><variable\_name>=<value> ....

To create environment variables at run time

- using -e or --env option at the runtime we can create env variables

- For multiple variables use multiple -e

ex: docker run .... -e <variable\_name>=<value> -e <variable\_name>=<value> ....

The best way to load multiple env variable is using env file

using --env-file <file\_path> at the runtime (with docker run command) we can load the env file containing n number variables.

### ARG

Using ARG we can pass parameters to Dockerfile as user inputs

ARG <var\_name>=<default\_value>

To pass the value at build time

docker build --build-arg <var\_name>=<value> ....

### WORKDIR

This is used to set the working directory for all the instructions that follows it

such as RUN, CMD, ENTRYPOINT, COPY, ADD

WORKDIR <path>

### EXPOSE

- used to expose a port to the docker network

- All containers in the same network will have access to exposed port.

EXPOSE <port\_number>

## Docker Volumes

  - As the layers inside the image are read-only which means once the image is created

  we cannot change/edit so we cannot put the container data in image.

- Container create a top most Read Write layer and all the runtime data is saved here.

- Container layer is temporary layer, if we lose the container, we lose data. so

  to retain/persist the container runtime data we need docker volumes.

### Bind Mounts

- we can mount host machine filesystem (files and directories) to the container

docker run -v <host\_path>:<container\_path>

### Docker Volumes

- These are docker managed filesystem and we use docker commands to manage these volumes

- Volumes are easier to manage, backup or migrate than bind mounts.

- Volumes supports many drivers which means we can mount many types of filesystems.

- Default location of docker volume is /var/lib/docker/volumes

docker run -v <volume\_name>:<container\_path>

To create volume

docker volume create <volume\_name>

To list volume

docker volume ls

To Delete volume

docker volume rm <volume\_name>

## Namespace

- Docker uses Linux namespaces to provide isolated workspace for processes called

  container

- when a container is created, docker creates a set of namespaces for it and provides

  a layer of isolation for container.

- Each container runs in a different namespace

namespace (To list - lsns)

## Cgroups

- Linux OS uses cgroups to manage the available hardware resources such as

  CPU, RAM, Disk, I/O.

- we can control the access and also, we can apply the limitations

To list - lscgroup

pid - namespace for managing processes (process isolation)

user - namespace for non-root user on linux.

uts - namespace for unix timesharing system which isolates kernel and version identifiers,

  time bonding of process.

ipc - (interprocess communication) namespace for managing the process communication.

mnt - namespace for managing filesystem mounts.

net - namespace for managing network interfaces.

## Docker networking

### Publish

PUBLISH = Expose + outside world port mapping

- publics will bind the container port to the host port which we can access from

  outside world using <host\_ip>:<port\_mapped>

- To publish a port

docker run -p <host\_port>:<container\_port> .....

-P publish\_all

It binds all the exposed ports of the container to host machine port

To map direct IP address to the host

port to port

ip:<host\_port>:<container\_port>

ip::<container\_port>

Range of ports

many to one

-p 8080-8090:8080

many to many

-p 8080-8085:8086-8091

- The total number of host ports in range should be same as the

  total number of container ports range.

## Docker network types

### 1. Bridge

- This is a private internal network created by docker on the host machine

  by name docker0

- This is the default network type for all the container which are created

          without any network configurations.

- By default, all the containers in the same bridge can communicate with

  each other without any extra configuration.

- We cannot use container name for communication only IP address is allowed in default bridge.

### Custom bridge

To create bridge network

docker network create --driver bridge my\_bride

- In custom bridge containers can communicate with each other with container name and also with IP address.

### 2. Host

- This driver removes the network isolation between docker and the host.

- The containers are directly connected to host machine network without

  extra layer of any docker network.

- Shares the same TCP.IP stack and same namespace of host machine.

- All the network interfaces which are there in host machine are

  accessible by this container.

### 3. None

- Containers are not attached to any network by docker.

- All the required network configurations need to be done

  manually.

- The host or any other containers won't be able to communicate

  with this container until a custom network is configured.

## Docker Architecture

### Docker Daemon

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

### The Docker client

The Docker client (docker) 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 dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

### Docker REST API

Docker provides an API for interacting with the Docker daemon

### Docker CLI

The CLI uses [Docker APIs](https://docs.docker.com/engine/api/) to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI

### Docker Objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

### Benifits of Docker

Flexible:

Complex applictions cab be divided and containerised in small compenets called microservice.

Lightweight:

Containers share the machine’s OS system kernel and therefore do not require

an OS per application, driving higher server efficiencies and reducing server and licensing costs

portable:

we can build images anywhere and then deploy to cloud, run anywhere.

## States of container / Lifecycle of container

1. Created - if container is newly created and container is not yet started.

2. Running - A currently running container. It means there is no problem

with container to run the process.

3. Exited - A container ran and completed ot executiom with failure.

4. paused - A container whose process have been paused. (we can unpause the container)

5. Dead - if docker daemon tried and failed to stop a container (host ram full)

6. Restarting - container will be in the phase of retarting the main process.

## Multistage build

How to optimize docker build process ?

How to reduce the size of the docker image or cotainer ?

After docker 1.6 version docker released this option.

1. There are 2 problems with the normal build process

1. Size: challenge is to keep the image and its containers size as minimal as possible.

2. larger the surface area more the application is vurnalable to attacks.

- Multistage build allows us to define multiple FROM in same Dockerfile.

- Dependency between multile FROM is maintained by naming FROM using

  AS keyword and we can refer this name in another FROM.

FROM <base\_image> AS <STAGE\_NAME>

- Only the final FROM image is created leaving back all the other FROM.

- Copy only the required files from the named FROM stage like below.

FROM final\_build

COPY --from <STAGE\_NAME> <src\_named\_stage> <dest>

2. Always try to use the slim / alpine / stretch version of base image instead

  od using the fully loaded base image.

### Advantages

* Make intermediate image layers shareable
* Keeps final images small and docker file readable
* Keep your secret safe
* Simplifies the images stages instantly

Example MultiStage Build: https://github.com/jaintpharsha/docker\_multi\_build.git

FROM node:10 AS ui-build

WORKDIR /usr/src/app

COPY WebApp/ ./WebApp/

RUN cd WebApp && npm install @angular/cli && npm install && npm run build

FROM node:slim AS server-build

WORKDIR /root/

COPY --from=ui-build /usr/src/app/WebApp/dist ./WebApp/dist

COPY package\*.json ./

RUN npm install

COPY index.js .

EXPOSE 3070

ENTRYPOINT ["node"]

CMD ["index.js"

## Docker-compose  Installation

1.sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2. sudo chmod +x /usr/local/bin/docker-compose

3. sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

**Docker-compose** is a tool for defining and running multiple container docker application with

  a single command.

- We use YAML file to do docker related configurations then with a single command

  we can execute this YAML file to create docker objects defined in this file.

docker-compose.yml

version: "3.8"

services:

jenkins:

image: jenkins/jenkins:lts

container\_name: dc-jenkins

ports:

- "8080:8080"

- "5000:5000"

networks:

- my\_brid

alpine:

build: .

# \* How multistage build works?

Within a docker file we can use multiple FROM statements, in an each FROM statement we can use different base image and each FROM specifies new stage to build.

We can selectively copy the artifacts from one stage to another stage leaving behind that we don’t want in the final stage.

- we use --copy - from to grab the required artifact from the previous FROM

we use. As parameter alias to get the dependency from one stage to another Stage

After all these stages the final / latest from will be executed to build an image

# How do you increase the size of the docker container size?

1. Stop the docker daemon after take the backup of the container

2. modify the docker config fill in /etc/sysconfig/docker-storage



3. in docker Storage option, specify the memory that you need to Increase.

4.Restart the service

# Docker File vs Docker Compose File

A Docker File is a simple text file that contains the commands a user could call to assemble an image

Docker Compose is a tool that allows to define and run multi-container Docker Applications