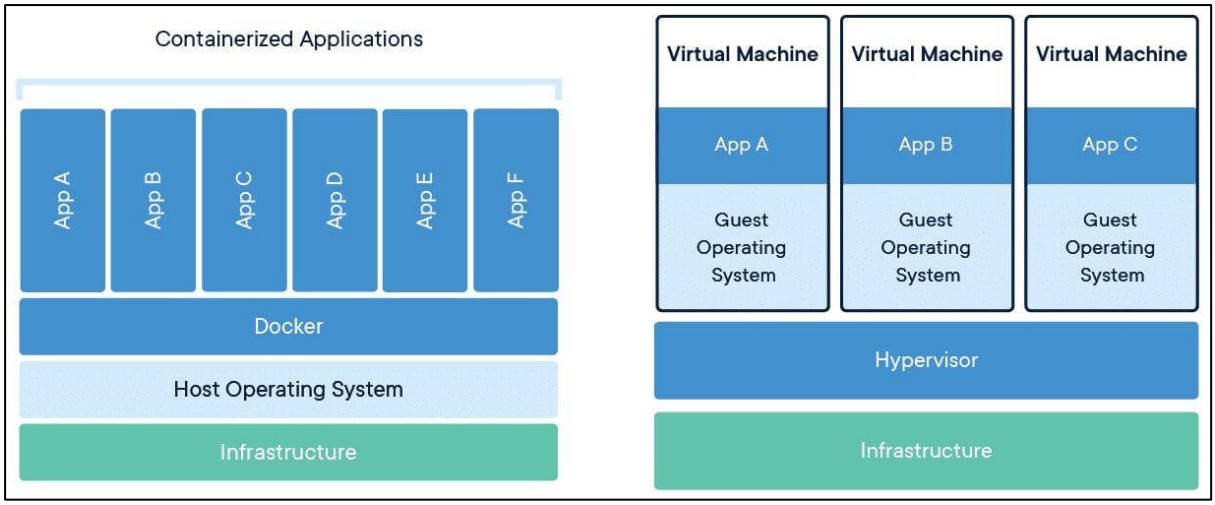


Basics of Docker:

1. Docker was first released in march 2013. It is developed by Solomon Hykes and Sebastien Paul.
2. Docker is an open-source centralized platform, designed to create, deploy, and run applications.
3. Docker uses the container on the host O.S. to run applications. It allows applications to use the same Linux kernel as a system on the host computer, rather than creating a whole virtual O.S.
4. We can install docker on any O.S. but the docker engine runs natively on Linux distribution.
5. Docker is written in the "GO" language.
6. Docker is a tool that perform OS-level virtualization, also known as containerization.
7. Before docker, many users face the same problem that a particular code runs in the developer's system but not in the user's system.

Advantages of Docker:

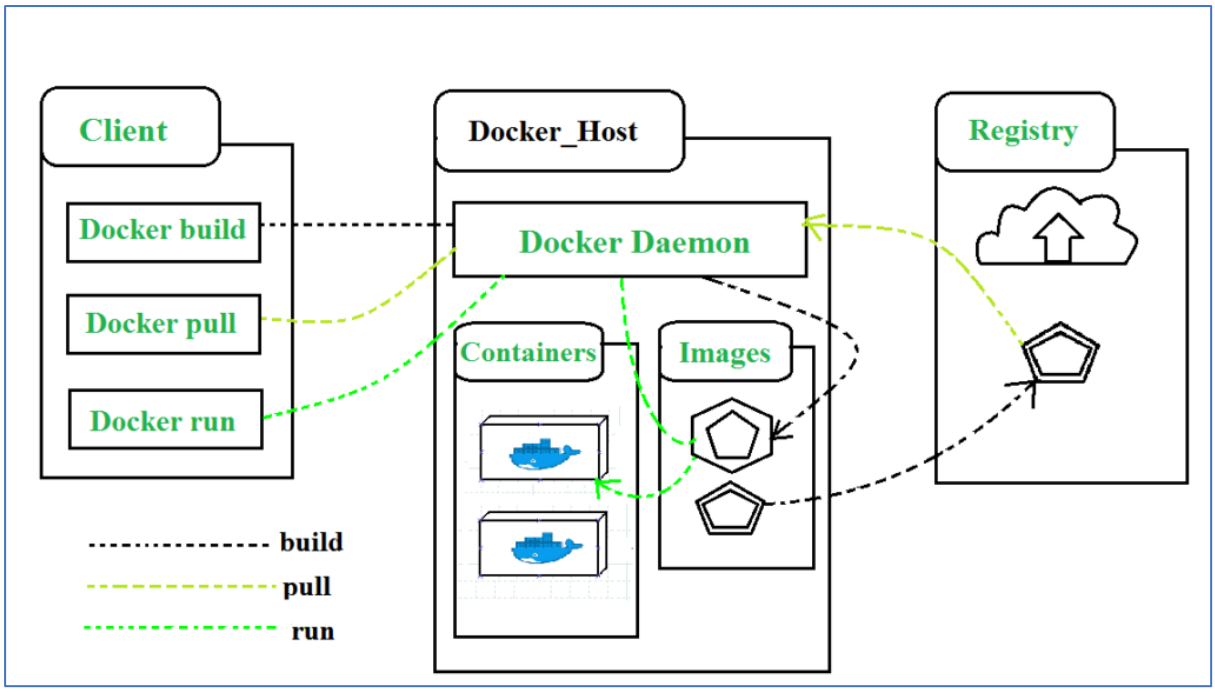
1. No pre allocation of RAM.
2. Less cost.
3. It is light in weight.
4. It can run on physical hardware, virtual hardware or on cloud.
5. It takes very less time to create the container.
6. Docker enables you to build a container image and use that same image across every step of the deployment process.
7. You can re-use the image.

Disadvantages of Docker:

1. Docker is not a good solution for application that requires a rich GUI.
2. Difficult to manage a large number of containers.
3. Docker does not provide cross platform compatibility, means if an application is designed to run in a docker container on windows, then it cannot run on Linux.

4. Docker is suitable when the development OS and testing OS are same, if the OS is different, we should use virtual machines.
5. No solution for data recovery and backup.

Note: when a image is running we can say it a container, when we send a container or non-running state, we can say it image.



Components of Docker:

Docker daemon/docker engine:

1. Docker daemon runs on the Host OS.
2. It is responsible for running container, and managing docker services.
3. Docker daemons can communicate with other daemons.

Docker client:

1. Docker users can interact with the docker daemon through a client.
2. Docker client uses commands and rest API to communicate with the docker daemon.
3. When a client runs any server command on the docker client terminal, the client terminal sends the docker commands to the docker daemon.

Docker Host:

1. Docker host is used to providing an environment to execute and run applications. It contains the docker daemon, images, containers, networks and storage.

Docker Hub/Registry:

1. Docker registry manages and stores the docker images.
2. There are two types of registries in the docker –

- Public registry: it is also called Docker hub.
- Private registry: it is used to share images within the enterprise.

Docker images:

1. Docker images are the read only binary templates used to create docker containers.
   Or
   A single file with all the dependencies and configurations required to run a program.

Ways to create an image:

- Take an image from docker hub.

- Create an image from Docker file.

- Create an image from existing docker containers.

Docker container:

1. The container holds the entire package that is needed to run an application.
   Or
   In other words, we can say that the image is a template and the container is a copy of that template.
2. Container is like virtualization when they run on the docker engine.
3. Images become containers when they run on the docker engine.

Docker installation and important commands:

1. To install docker on Ubuntu

- sudo apt-get install docker.io -y

[sudo usermod -aG docker $USER]

2. To check whether the service is started or not-

- service docker status

3. To start the service

- service docker start

4. To see all the images present in your local-

- docker images

5. To find out images in the docker hub-

- docker search <image_name>

6. To download an image from docker-hub to our local machine-

- docker pull <image_name>

7. To start/stop the container.

- docker start/stop <container_name or ID>

8. To go inside the container

- docker attach <container_name or ID>

9. To see all the containers

- docker ps -a

10. To see only running containers-

- docker ps

11. To delete the container-

- docker rm <container_name or ID>

12. To delete the images-

- docker rmi <image_name>

13. To exit from the container-

- exit

14. To give a name to a container and run it –

- docker run -it –name <container_name> <image_name> /bin/bash

[it – interactive mode and direct to terminal]

15. To create the image from the container-

- docker commit <container_name> <container's_image_name>

Pull an image from the docker hub and create one container and go inside that container and create a normal demo file there.

- docker diff <container_name> : this command will tell you the difference between the base image and the changes you made on it.

Docker image creation using Dockerfile:

Dockerfile:

- Dockerfile is a text file, it contains some set of instruction.

- Automation of docker image creation.

FROM:

- for the base image. This command must be on the top of the Dockerfile.

RUN:

- to execute commands, it will create a layer in the image.

MAINTAINER:

- Author/owner/Description

COPY:

- Copy files from the local system. {we need to provide a source, and destination.}

[we cannot download the file from the internet or any remote repo]

ADD:

- Similar to copy, it provides a feature to download files from internet, also we extract the file from the docker image side.

EXPOSE:

- To expose the ports, such as port 80 for nginx and port 8080 for tomcat etc.

WORKDIR:

- To set a working directory for a container.

CMD:

- Execute commands but during container creation.

ENTRYPOINT:

- Similar to CMD, but has higher priority over CMD, the first commands will be executed by ENTRYPOINT only.

ENV:

- Environment variables.

ARG:

- To define the name of a parameter and its default value, the difference between ENV and ARG is that, after you set on env. using ARG, you will not be able to access that later on when you try to run the docker container.

Creation of Dockerfile:

- Create a file named “Dockerfile”.

- Add instruction in the Dockerfile.

- Build Dockerfile to create an image. {**docker build -t .**}

- Run image to create the container.

```
#get the base image
FROM openjdk:11

# Create a working directory to keep all the files
WORKDIR /app

#code copy to the image for running in container
COPY Hello.java /app

#compile the code
RUN javac Hello.java

#now app is ready to run

#actually passing the run commands as arguments
CMD ["java","Hello"]
```

Docker Hub:

It is a public repository, same like git hub, where we can push and pull the images.

- so that we can push our images and use it from anywhere.

ECR (Elastic container registry) – same like dockerhub, we use this in AWS

- docker login

Fill the username and password there.

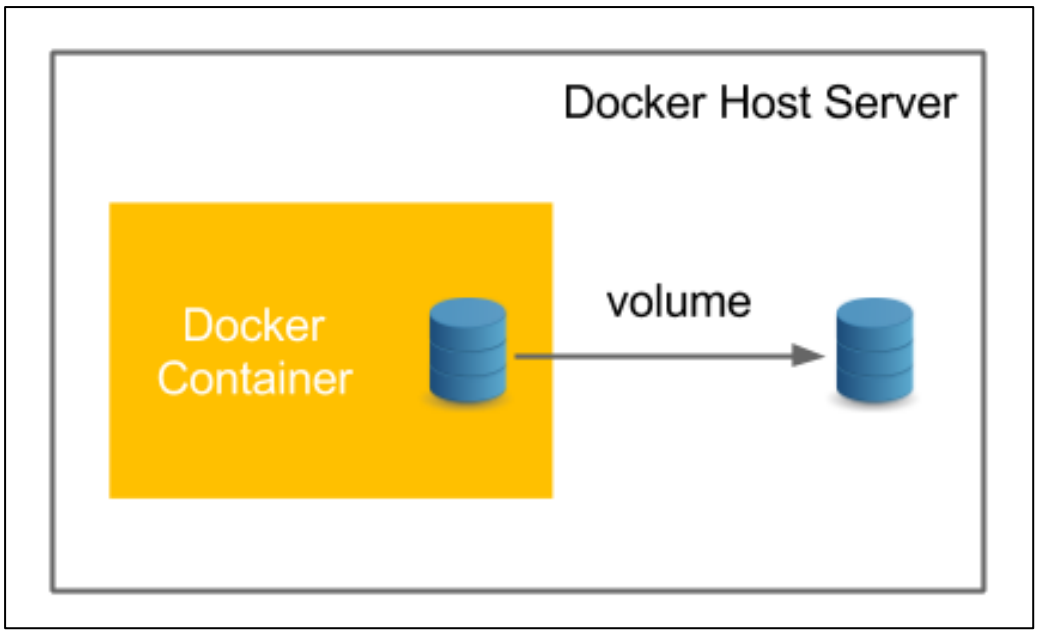
- docker tag <old_image_name> <username/new_image_name>

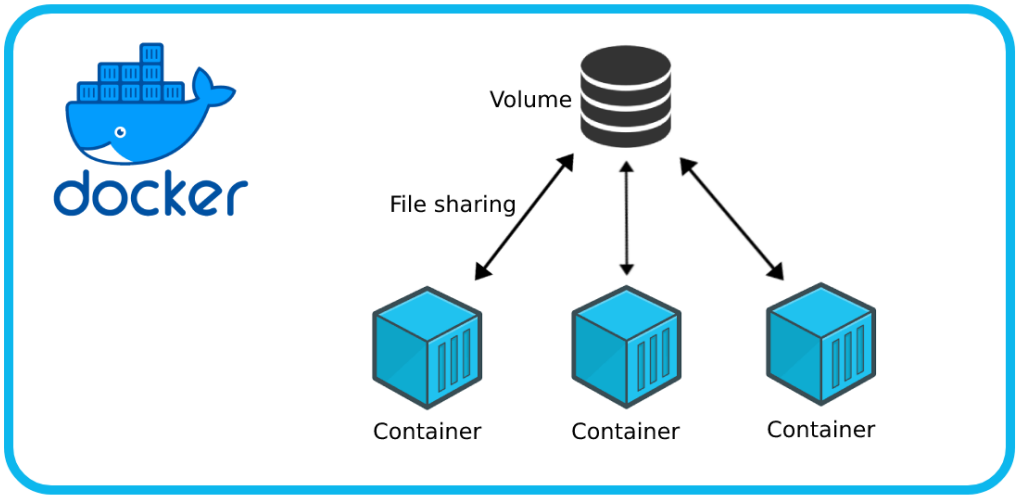
- docker tag - we use it to rename image. (we use our dockerhub_username/image_name – only after we can push it to the docker hub)

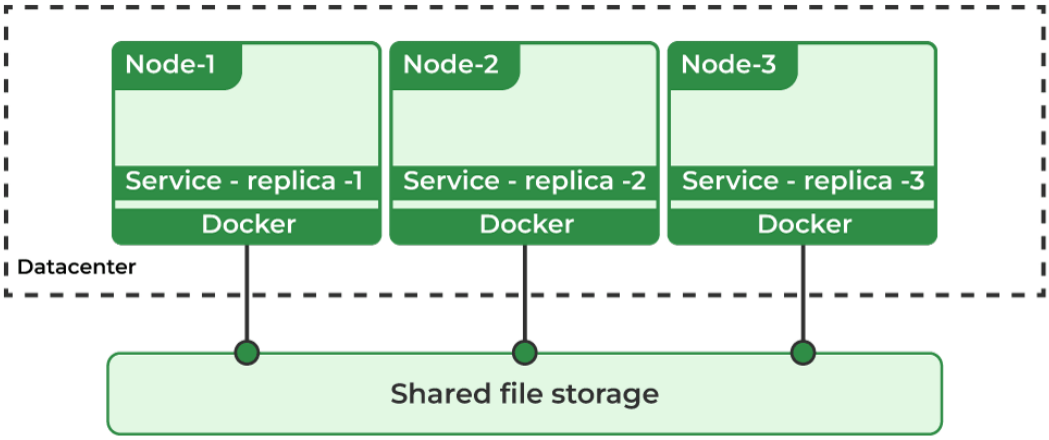
- docker push username/image_name

Docker Volume:

A Docker volume is an independent file system entirely managed by Docker and exists as a normal file or directory on the host, where data is persisted.







1. Docker volume is simply a directory inside our container.

2. Firstly, we have to declare this directory as a volume and then share the volume.
3. Even if we stop the container, still we can access volume.
4. The volume will be created in one container.
5. You can declare a directory as a volume only while creating the container.
6. You can't create volume from the existing container.
7. You can share one volume across any number of containers.
8. The volume will not be included when you update an image.
9. We can map volume in two ways –
   - Container ←→ Container
   - Host ←→ Container

Benefits of Volume:

- Decoupling container from storage.

- Share volume among different containers.

- Attach the volume to containers.

- On deleting the container, the volume does not delete.

- docker volume ls :- to see all the existing volumes.

Create one directory named volumes (not inside your git clone directory)

To create a docker volume----------

- docker volume create --name <volume_name> --opt type=none --opt device=</home/ubuntu/volumes/path> --opt o=bind

To run the container and mount that created volume----------

- docker run -d -p 8000:8000 --mount source=<volume_name>,target=/data <image-name>

(-p 8000:8000 ---- Mapping the port for Host and docker container)

- docker volume inspect <volume name>

- docker system prune

Docker network: so that two containers can communicate with each other

First create a network (bridge_type) so that we can add the containers in the network.

-- docker network ls

-- docker network create -d bridge <nw_name>

To create a mysql database container & attach that docker network(bridge)-------

-- docker run -d --name <mysql> -p 3306:3306 -e MYSQL_ROOT_PASSWORD=test@123 -e MYSQL_DATABASE=testdb -e MYSQL_USER=admin -e MYSQL_PASSWORD=root --network <network(bridge)_name> mysql:latest

To verify if that container attached inside the docker network---

-- docker inspect <network(bridge)_name>

To create another container inside the same network and attach it to the database----------

-- docker run -d --name <container-name > -p 5000:5000 -e MYSQL_HOST=<mysql(container-name)> -e MYSQL_USER=admin -e MYSQL_PASSWORD=root -e MYSQL_DB=testdb --network <network(bridge)_name> <image-name>

Again –To verify -- docker inspect <network(bridge)_name>

To go inside the mysql container && some more commands------

-- docker exec -it <container_name/ID> bash

-- mysql -u root -p (put the password)

-- show databases;

-- use database-name;

-- show tables;

-- select * from table-name;

To create a table -----

-- CREATE TABLE table-name (

id INT AUTO_INCREMENT PRIMARY KEY,

message TEXT

);

DOCKER-COMPOSE:

- Docker Compose is a tool that helps you define and manage multi-container Docker applications. In simpler terms, it allows you to describe how different services (or containers) in your application should run, connect, and interact with each other, all in a single file.

- Docker Compose helps you manage these multi-container applications by defining them in a single configuration file.

- The configuration for Docker Compose is typically stored in a file called docker-compose.yml. This file describes how your services should behave, what images to use, how they should connect, and any other configurations needed.

- Docker Compose automatically creates a network for your services, allowing them to communicate with each other.

- You can also specify volumes in your Docker Compose file, which allows you to persist data outside of the containers. This is useful for databases and other applications that need to store information.

- **docker-compose up -d**: starts the containers defined in your configuration,

- **docker-compose down**: stops and removes them.

Example: -

```yaml
version: '3'
services:

  backend:
    build:
      context: .
    ports:
      - "5000:5000"
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: admin
      MYSQL_PASSWORD: admin
      MYSQL_DB: myDb
    depends_on:
      - mysql

  mysql:
    image: mysql:5.7
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: myDb
      MYSQL_USER: admin
      MYSQL_PASSWORD: admin
    volumes:
      - ./message.sql:/docker-entrypoint-initdb.d/message.sql   # Mount sql script into container's /docker-entrypoint-initdb.d directory to get table automatically created
      - mysql-data:/var/lib/mysql  # Mount the volume for MySQL data storage

volumes:
  mysql-data:
```