**DOCKER**

**Application Development**

* Collection of programs is called as software project
* Software project contains several components

1) Front end components (User interface logic)

2) Backend components (Business Logic)

3) Database Components (Persistence Logic)

* In order to deploy our application in a machine we need to setup all the Softwares which are required to our application

Ex: OS, Java 1.8v, MYSQL DB, Tomcat Web Server 9.0v etc.....

* In Realtime project should be deployed into multiple environments for testing purpose

Ex : DEV, SIT, UAT, PILOT and PROD

* DEV env will be used by Developers to perform integration testing
* SIT env will be used by Testing team to test functionality of the application
* UAT env will be used by Client to test functionality of the application
* PILOT env means pre-production testing env
* PROD means live environment (It is used to deliver the project)
* To deploy application to these many enivornments we need to take of all the softwars required to run our application in all environments. It is very difficult task.

**Virtualization**

* Installing Multiple Guest Operating Systems in one Host Operating System
* **Hypervisior** S/w will be used to achieve this
* We need to install all the required softwares in HOST OS to run our application
* It is old technique to run the applications
* System performance will become slow in this process
* To overcome the problems of Virtualization we are going for Containerization concept

**Containerization**

* **It is used to package all the softwares and application code in one container for execution**
* Container will take care of everything which is required to run our application
* We can run the containers in Multiple Machines easily
* **Docker is containerization software**
* Using Docker **we will create container** for our application
* Using Docker **we will create image** for our application
* Docker images we can share easily to mulitple machines
* Using Docker image we can create docker container and we can execute it

**Conclusion**

Docker is containerization software

Docker will take care of application and application dependencies for execution

Deployments into multiple environments will become easy if we use Docker containers concept

**++++++++Install Docker in Amazon Linux++++++++++++**

* **$ sudo yum update -y**
* **$ sudo yum install docker -y**
* **$ sudo service docker start**

# add user to docker group by executing below command

* **$ sudo usermod -aG docker ec2-user**
* **$ docker info**

#Restart the session

* **$ exit**

Then press 'R' to restart the session (This is in MobaXterm)

**Docker Commands**

See docker info

* **$ docker info**

To see docker images execute below command

* **$ docker images**

Pulling hello-world docker image

* **$ docker pull hello-world**

See docker image

* **$ docker images**

Running hello-world docker image

* **$ docker run hello-world**

See docker container

* **docker ps -a**

To remove docker image

* **$ docker rmi <image\_id>**
* **$ docker rmi -f <image\_id>**

To remove docker container

* **$ docker rm <container\_id>**
* **$ docker rm -f <container\_id>**

To run docker container

* **$ docker run -d -p (Host machine port)80:80(container port) <image\_name>**
* **$ docker run --name=firstimage -d -p 80:80 nginx**

**Note: Create account in Docker Hub (https://hub.docker.com/)**

**Dockerfile**

Dockerfile is file which contains instructions to create an image which contains Docker Domain Specific Key Words to build image.

**DockerImage**

It's a package which contains everything (Softwares+ENV+Application Code) to run your application.

**DockerContainer**

Run time instance of an image. If you run docker image container will be created that's where our application (process) is running.

**DockerRepo/Registry**

We can store and share the docker images.

**Public Repo**

Docker hub is a public repository which contains all the open source softwares as a docker images. We can think of docker hub as play store for docker images.

**Private Repo**

(Nexus, JFrog, D.T.R(Docker Trusted Registory)),

**AWS ECR**

We can store and share the docker images with in our company network using private repo

**Docker Engine/Daemon/Host**

It's a software or program using which we can create images & contianers.

Docker is cross platform.

Docker CE

Docker CE will not be supported by Redhat.

Docker EE

Docker EE will be support most of the os including redhat.

**What is docker hub?**

It's a public repository for docker images.

You can think as play store for docker images.

Create docker file with below content

* **$ vi Dockerfile**

FROM ubuntu

RUN echo "Run One Updated"

RUN echo "RUN TWO"

CMD echo "Echo From Image"

CMD echo "Echo From Latest"

RUN echo "RUN Three"

Login into Dockerhub account

* **$ docker login**

Build docker image using docker file

* **$ docker build -t(filename) imageone . (dot represent search for current location)**

To save docker image in docker hub Tag is required

* **$ docker tag imageone ashokit/ imageone**

Push docker image

* **$ docker push ashokit/ imageone**

**Dockerfile**

* **Dockerfile contains instructions to build docker image**
* In Dockerfile we will use DSL (**Domain Specific Language**) keywords
* Docker engine will process Dockerfile instructions from top to bottom
* Below are the Dockerfile keywords

FROM

MAINTAINER

COPY

ADD

RUN

CMD

ENTRYPOINT

ENV

LABEL

USER

WORKDIR

EXPOSE

VOLUME

**1. FROM**: It indicates base image to run our application. On top of base image we will create our own image

**Syntax: FROM <IMAGE-NAME>**

Example:

FROM java:jdk-1.8.0

FROM tomcat:9.2

FROM mysql

**2. MAINTAINER:** It represents who is author of Dockerfile

Ex: MAINTAINER Ashok <ashokitschool@gmail.com>

**3. COPY:** It is used to copy files / folders to image while creating an image

**Syntax : COPY <source> <destination>**

Example :

copying war file from target directory to tomcat/webapps directory

COPY target/maven-web-app.war /usr/local/tomcat/webapp/maven-web-app.war

**4. ADD**

ADD is also used to copy files to image while creating an image

ADD keyword can **download files** from remote location (http)

ADD keyword will extract tar file while copying to imgae

Note: zip files we have to extract manually

Syntax:

**ADD <source> <destination>**

**ADD <url-to-download> <destination>**

Q) What is the difference between COPY and ADD?

**5. RUN**

It is used to execute commands on top of base image

**Run command instructions will execute while creating an image**

We can write multiple RUN instructions, they will execute in the order (from top to bottom)

Example:

RUN mkdir workspace

RUN yum install git

**6. CMD**

CMD is also used to execute commands

**CMD instructions will execute while creating container**

Example:

CMD sudo start tomcat

We can write multiple CMD instructions in Dockerfile but Docker will process **only last CMD instruction.**

Note: There is no use of writing multiple CMD instructions in Dockerfile

**Sample Dockerfile**

FROM ubuntu

MAINTAINER Ashok IT

RUN echo "Run One"

RUN echo "Run Two"

CMD echo "CMD One"

CMD echo "CMD Two"

RUN echo "Run Three"

Build image using docker file

* **$ docker build -t imageone .**

Run image

* **$ docker run imageone**

**Note: CMD instruction we can override using runtime CMD**

It will print only date (CMD will not execute)

* **$ docker run imageone date**

We can change docker file name

* **$ mv Dockerfile Dockerfile\_One**

Creating Docker image using Dockerfile\_One

* **$ docker build -f Dockerfile\_One -t imagetwo** .

**7. ENTRYPOINT:** ENTRYPOINT instructions will execute **while creating container**

**Note: CMD instructions we can override where as ENTRYPOINT instructions we can't override**

Example:

ENTRYPOINT [ "echo", "Welcome to Ashok IT "]

**8. WORKDIR:** It is used to set Working Directory for an image / container

Ex: WORKDIR <DIR-PATH>

**Note: The Dockerfile instructions which are available after WORKDIR those those instructions will be process from given working directory**

**9. ENV**

ENV is used to set Environment Variables

Ex: ENV <key> <value>

**10. LABEL**

LABEL will represent data in key value pair

It is used to add meta data for our image

**Ex: LABEL branchName release**

**11. ARG**

It is used to avoid hard coded values in Dockerfile

Ex:

ARG branch=develop

LABEL branch $branch

**Note: We can pass argument values in RUNTIME**

**$ docker build -t imageone --build-arg branch=feature**

**12. USER:** We can set user for an image / container

Note: After USER instruction, remaining instructions will be processed with given USER

**13. EXPOSE:** It represents on which **port number** our container is running

It is just like a documentation to understand container running port number

**14. VOLUME:** It is used for **data storage**

**Dockerizing Spring Boot Application**

FROM java:8-jdk-alpine

COPY ./target/spring-boot-docker-app.jar /usr/app/

WORKDIR /usr/app

ENTRYPOINT ["java","-jar","spring-boot-docker-app.jar"]

**Dockerizing Java Web App**

FROM tomcat:8.0.20-jre8

COPY target/java-web-app\*.war /usr/local/tomcat/webapps/java-web-app.war

Note: After running the container access application using below URL

URL : http://ec2-vm-public-ip:8080/java-web-app/

**Dockerizing Python Flask Application**

FROM python:3.7

WORKDIR /opt/app

COPY . .

RUN pip install --no-cache-dir -r requirements-prod.txt

EXPOSE 5000

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]

To display all running containers

* **$ docker ps**

To display all containers

* **$ docker ps -a**

Stop the container

* **$ docker stop <container-id>**

Remove the container

* **$ docker rm <container-id>**

Remove stopped container & unused images

* **$ docker system prune -a**

**Docker Volumes**

* Docker volumes are used to **persist the data generated by Docker containers**
* Using Docker volume we can de-couple data storage from Docker container
* Using Docker volume we can **share the data among multiple Docker containers**
* **On deletion of Container, Docker Volume will not be deleted.**

**Docker Volume Commands**

To display docker volume commands

* **$ docker volume --help**

Create Docker Volume

* **$ docker volume create <volume-name>**

Display Docker volumes

* **$ docker volume ls**

Inspect the volume

* **$ docker volume inspect <volume-name>**

Remove docker volume

* **$ docker volume rm <volume-name>**

Rmove unused volumes

* **$ docker volume prune**

**Pulling nginx image from dockerhub**

* **$ docker pull nginx**

Running nginx image using docker container with container name as "webapp1"

* **$ docker run --name=webapp1 -d -p 80:80 nginx**

Note: Access static website using EC2 VM public ip which is hosted by nginx

Modify static website content in index.html file (file is available in container)

* **$ docker exec -it webapp1 bash**
* **$ cd /usr/share/nginx/html**
* **$ echo "Welcome to Ashok IT" > index.html**

Note: Access static website using EC2 VM public ip which is hosted by nginx (Modified content shud display here)

Stop docker container which is "webapp1"

* **$ docker stop <container-id>**

Start another docker container with name as "webapp2"

* **$ docker run --name=webapp2 -d -p 80:80 nginx**

Note: Access static website using EC2 VM public ip which is hosted by nginx

(Changes we made in first container will not reflect here)

Stop docker container which is "webapp2"

* **$ docker stop <container-id>**

Create docker volume

* **$ docker volume create new\_vol**

Start docker container by mouting docker volume

* **$ docker run -d --name=webapp20 --mount source=new\_vol,destination=/usr/share/nginx/html -p 80:80 nginx**

Note: Access static website using EC2 VM public ip which is hosted by nginx

* **$ docker exec -it webapp20 bash**
* **$ cd /usr/share/nginx/html**
* **$ echo "Welcome to Ashok IT" > index.html**

Note: Access static website using EC2 VM public ip which is hosted by nginx

Stop the docker container

* **$ docker stop <container-id>**

Run another docker container (modified data should reflect here also)

* **$ docker run -d --name=webapp21 --mount source=new\_vol,destination=/usr/share/nginx/html -p 80:80 nginx**

**Host Directory Mounting to Docker Container**

Create Directory in host machine

* **$ mkdir /tmp/nginx/html**

Run Docker container for nginx image with directory mouting (container name is C1)

* **$ docker run -d -p 80:80 -v /tmp/nginx/html:/usr/share/nginx/html --name c1 nginx:latest**

See the containers which are running (c1 should be running)

* **$ docker container ls**

Note: Acess Nginx webserver using Host Machine Public IP

(It will not displayed bcz index.html file not available in container)

Goto Mounted directory in host machine

* **$ cd /tmp/nginx/html**

Create index.html file

* **$ vi index.html**

Note: write the content in index.html file then save it close it

Access Nginx webserver in browser (Changes will be reflected)

Note: We have created index.html file in host machine mounted directory it is reflecting in container

**Docker Compose**

* In Realworld applications are getting developed using Microservices Architecture
* In Microservices architecture several APIs (**Application program interface**) will be available
* Every API is a project and it should run in a container
* Running multiple containers manually for all APIs is difficult job

Note: To solve this problem Docker Compose came into picture

* Docker compose is a tool which is used to manage multi container based applications
* Using Docker compose we can define & deploy multi container based applications easily
* We will give input to docker compose tool **using YAML file** to run multiple containers
* Docker Compose YML file should have information related to all our service

**Sample Docker Compose YML (docker-compose.yml)**

version:

services:

network:

volumes:

Docker Compose Default File Name: docker-compose.yml

Create and start the containers

* **$ docker-compose up**

List Docker containers started by docker compose

* **$ docker-compose ps**

Stop and remove containers

* **$ docker-compose down**

List down running container images

* **$ docker-compose images**

Using different file

* **$ docker-compose -f <filename> up**

**Docker Compose Setup**

Download docker compose

**$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**

Give permission

**$ sudo chmod +x /usr/local/bin/docker-compose**

How to check docker compose is installed or not

**$ docker-compose --version**

Create a docker compose file for setting up dev environment. Mysql container is linked with wordpress container.

**$ vim docker-compose.yml**

---

services:

mydb:

environment:

MYSQL\_ROOT\_PASSWORD: ashokit

image: "mysql:5"

mysite:

image: wordpress

links:

- "mydb:mysql"

ports:

- "9090:80"

version: "3"

Lets remove all the running container

**$ docker rm -f $(docker ps -aq)**

How to start the above services from yml file

**$ docker-compose up**

We got lot of logs coming on the screen. to avoid it we use -d option

**$ docker-compose stop**

Remove the container

**$ docker rm -f $(docker ps -aq)**

Run containers in detached mode

**$ docker-compose up -d**

To check wordpress site

**URL: http://public\_ip:9090**

To stop both the containers

**$ docker-compose stop**

**Spring Boot App with MSQL DB using Docker Compose**

To run spring boot application we will use below Dockerfile

Dockerfile

----------------

FROM openjdk:jdk1.8

COPY target/sb-app.jar /usr/local/sb-app.jar

WORKDIR /usr/local/

EXPOSE 8080

ENTRYPOINT [ "java" , "-jar", "sb-app.jar" ]

If we are using MySQL DB in spring boot application then we need to MySQL DB in a container

Note: Instead of managing two containers separately we can use Docker Compose.

**Docker Compose Configuration (docker-compose.yml)**

version: "3"

services:

boot-app:

image: sb-rest-api

ports:

- "8080" : 8080

depends\_on:

- mysql-db

mysql-db:

image: mysql:8

environment:

- MYSQL\_ROOT\_PASSWORD = ashokit

- MYSQL\_DATABASE = bootdb

**Docker Network**

* Networking is all about communication among processes
* Docker networking enables a user to link a docker container to as many networks as they require
* Docker network is used to provide complete isolation for Docker containers

**Advantages of Docker Networking**

1) They share single operating system and maintain containers in isolated manner

2) It requires fewer OS instance to run the workload

3) It helps in fast delivery of the software

4) It helps in application portability

When Docker s/w is installed in a machine by default 3 docker networks will be configured

**1) none**

**2) host**

**3) bridge**

Note: One container we can attach to multiple networks

When container is attached to multiple networks then those containers can communicate

Docker providing Networking to containers using Network Drivers

**Docker Network Drivers**

1) Bridge

2) Host

3) None

4) Overlay

5) Macvlan

**Bridge Driver**

Bridge: This the default network driver created on the Docker host machine

Note: Bridge network drivers are very useful **when application running in standalone container**

We can see more details about bridge driver using below command

**$ docker network inspect bridge**

**Host Driver**

It is useful when standalone container is available

**The container will not get any IP address when we enable Host Driver**

Note: For example a container is executed that binds to port 80 with Host Network Driver. In this case we no need to map container port to host machine port.

This is useful when we are running our containers with large no. of ports

**None Driver**

In this type of network, the containers will have no access to network

**Overlay Driver**

We will use Docker Swarm to orchestrate our Docker containers

Overlay Driver will be used when we have Docker Swarm cluster

**Macvlan Driver**

This network driver will assign a MAC address to a container

MAC address will make our device as Physical

Using this MAC address Docker engine routes the traffic to a particular route

Macvlan driver simplifies communication betweek containers

**Working with Docker Networking Commands**

To list all networks

**$ docker network ls**

Running docker container with default network

**$ docker run --name nginx -d -p 80:80 nginx**

**$ docker inspect nginx**

Creating our own bridge network

**$ docker network create --driver bridge my-network**

**Note:** If we don't specify driver then by **default it will take bridge driver**

Run the docker container using custom network which we have created

**$ docker run --name nginx20 -d --network my-network -p 7070:80 nginx**

Check the containers attached to my-network

**$ docker network inspect my-network**

Run 2 Docker containers using My-Network and check connectivity between 2 containers using ping command

**$ docker run --name nginx30 -d --network my-network -p 8080:80 nginx**

**$ docker run --name nginx31 -d --network my-network -p 9090:80 nginx**

Get IP address of Docker Container

Synax: **$ docker inspect -f '{{range.NetworkSettings.Networks}} {{.IPAddress}}{{end}}' <container-id-or-name>**

Nginx30 Container IP: 172.19.0.2

Nginx31 Container IP: 172.19.0.3

Connect to nginx30 container and ping nginx31 container IP using ping command

**$ docker exec -it nginx30 /bin/bash**

**$ apt-get update**

**$ apit-get install iputils-ping**

**$ ping 172.19.0.3**

Connect to nginx31 container and ping nginx30 container IP using ping command

**$ docker exec -it nginx31 /bin/bash**

**$ apt-get update**

**$ apit-get install iputils-ping**

**$ ping 172.19.0.2**

**Running Containers Using Host Network Driver**

When we use Host Network driver port mapping is not required

**$ docker run --name nginx35 --network host -d nginx**

Note: If you observer we are running nginx container without port mapping because we are using Host Network Driver

**Running Containers Using Macvlan Network Driver**

In Docker, a common question that usually comes up is “How do I expose my containers directly to my local physical network?” This is especially so when you are running monitoring applications that are collecting network statistics and want to connect container to legacy applications. A possible solution to this question is to create and implement the macvlan network type.

Macvlan networks are special virtual networks that allow you to create “clones” of the physical network interface attached to your Linux servers and attach containers directly your LAN. To ensure this happens, simple designate a physical network interface on your server to a macvlan network which has its own subnet and gateway.

# Get IP address of Docker Host Machine

$ ifconfig

Note : Take ipaddress from eth0

inet : 172.31.13.126

Subnet : 172.31.13.0/24

Gateway : 172.31.13.1

# Create macvlan network using subnet and gateway

$ docker network create -d macvlan \

--subnet=172.31.13.0/24 \

--gateway=172.31.13.1 \

-o parent=eth0 \

my-macvlan-network

# Check the network created

$ docker network ls

# inspect network created

$ docker network inspect my-macvlan-network

# Run docker container using macvlan network we have created

$ docker run --rm -itd \

--network=my-macvlan-network \

--ip=172.31.13.110 \

alpine:latest \

/bin/sh

# inspect network created

$ docker network inspect my-macvlan-network

What is Virtualization

What is Containeraization

What is Docker

Advantages of Docker

Docker Architecture

Docker Setup

Docker Terminology

Dockerfile

Docker image

Docker hub

Docker Container

Docker image commands

Docker container commands

Writing Dockerfile

SpringBoot App Dockerization

Java Web App Dockerization

Python Flask App Dockerization

Stateless Container Vs Statesfull Container

Docker Volumes

Named Volumes

Binded Mount Volumes

Docker Networking

Docker Network Driver

Creating Custom Networks

Running containers with Custom Network

**Docker Swarm**

* It is a container orchestration sofware
* Orchestration means managing processes/containers
* Docker Swarm is used to setup Docker Cluster (Load Balancing)
* Cluster means group of servers
* Docker swarm is embedded in Docker engine
* We will setup Master and Worker nodes using Docker Swarm cluster
* Master will schedule the tasks (containers) and manage the nodes and node failures
* Worker nodes will perform the action (containers will run here)

**Swarm Features**

1) Cluster Management

2) Decentralize design

3) Declarative service model

4) Scaling

5) Multi Host Network

6) Service Discovery

7) Load Balancing

8) Secure by default

9) Rolling Updates

**Setup**

Create 3 EC2 instances (ubuntu)

Note: Enable 2377 port for Swarm Cluster Communications

**$ curl -fsSL https://get.docker.com -o get-docker.sh**

**$ sudo sh get-docker.sh**

1 - Master

2 - Nodes

Connect to Master Machine and execute below command

**$ sudo docker swarm init --advertise-addr 172.31.42.162**

**$ sudo docker swarm join-token worker**

**$ sudo docker swarm join --token SWMTKN-1-21l3z1izmf6plkgprlzfcr87fcxcsl3k2k79iax7yfyh3k4a00-cfgwhikmbcgsklhsxkroj8vad 172.31.38.222:2377**

Q) what is docker swarm manager quarm?

Ans) If we run only 1 master node then we can't get gaurantee High Availability for the application

Formula : (n-1)/2

If we take 2 master nodes

2-1/2 => 0.5 ( It can't become master )

3-1/2 => 1 (it can be leader when the main leader is down)

Note: Always use odd number for Master machines when we are using Docker Swarm Cluster

If we use "docker run" command then our application will executed in one container

In Docker swarm we need to deploy our application as a service.

**Docker Swarm Service**

Service is collection of one or more containers of same image

There are 2 types of services in docker swarm

1) Replica (default mode)

2) global

**$ sudo docker service create --name <serviceName> -p <hostPort>:<containerPort> <imageName>**

**$ sudo docker service create --name java-web-app -p 8080:8080 ashokit/javawebapp**

Note: By default 1 replica will be created

Check the services created

**$ docker service ls**

We can scale docker service

**$ sudo docker service scale <serviceName>=<no.of.replicas>**

Inspect docker service

**$ sudo docker service inspect --pretty java-web-app**

See service details

**$ sudo docker service ps java-web-app**

Remove one node from swarm cluster

**$ sudo docker swarm leave**

Remove docker service

**$ docker service rm java-web-app**