Docker & Kubernetes

Application Tech Stack : It represents technologies used in the application

1) Frontend Stack : HTML, CSS, JS, BS & Angular / React JS

2) Backend Stack : Java / .Net / Python / Node JS

3) Database : Oracle / MySQL / PostgresSQL / Mongo DB

Docker

- Docker is a containizeration platform
- Docker is used to simplify application deployment process in Multiple Environments (DEV, SIT, UAT, PILOT and PROD)
- Docker is used to package application code + application dependencies for easy execution
- Using Docker we will create Docker images
- Docker Image contains App code + App dependencies
- We can run docker image in any machine. It will take care of dependencies & execution
- When we run Docker image, it will create Docker container
- Docker Container will run our application

Virtualization

- Installing Multiple Guest Operating Systems in one Host Operating System
- Hypervisions S/w will be used to achieve this
- We need to install all the required software's 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 a 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 a 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

Container = Application Code + Application Libraries + Application

- Docker is a containerization software
- Using Docker we will create Docker image

Docker Image = Application code + Application libs (maven dependencies) + Application Dependencies(java, tomact, mysql etc...)

- Once docker image is created then we can use jenkins to run docker image in multiple machines
- Jenkins is just deployment software. We will use jenkins to run docker images in all environments
- When we run Docker image it will create Docker container
- Docker Container means Runtime instance of our application

Docker Architecture

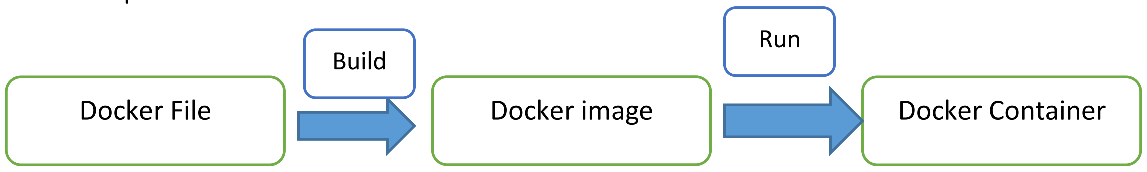
1) Dockerfile **2) Docker Image**

3) Docker Registry **4) Docker Container**

- Docker file contains set of instructions to build docker image. In dockerfile we will specify which software's are required to run our code/application.
- Docker image means a package which contains application code + app libs + app dependencies
- Docker Registry is a place which is used to store docker images for future purpose

Ex: Docker Hub, Amazon ECR etc....

- Docker container is runtime instance of our application. When we run Docker Image it will create Docker Container. Inside Container our application and application dependencies will be available.



Install Docker in Linux VM

- Loging into AWS account
- Create Linux Virtual Machine using Amazon Linux AMI
- Connect to Linux VM using MobaXterm
- Execute below commands to install Docker s/w

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

- # add ec2-user to docker group by executing below command (to give docker permissions to ec2-user accnt)

```
$ sudo usermod -aG docker ec2-user
```

- # Close the terminal

```
$ exit
```

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

- #execution below command to see docker status

```
$ docker info
```

Basic Docker Commands

- # display docker images available in our machine

```
$ docker images
```

- # download docker image

```
$ docker pull <image-name / image-id>
```

- # Run docker image

```
$ docker run <image-name / image-id>
```

- # Delete docker image

```
$ docker rmi <image-name / image-id>
```

- # Display all running docker containers

  $ docker ps

- # display all running and stopped containers

  $ docker ps -a

- # Delete docker container

  $ docker rm <container-id>

- # Delete docker image forcefully

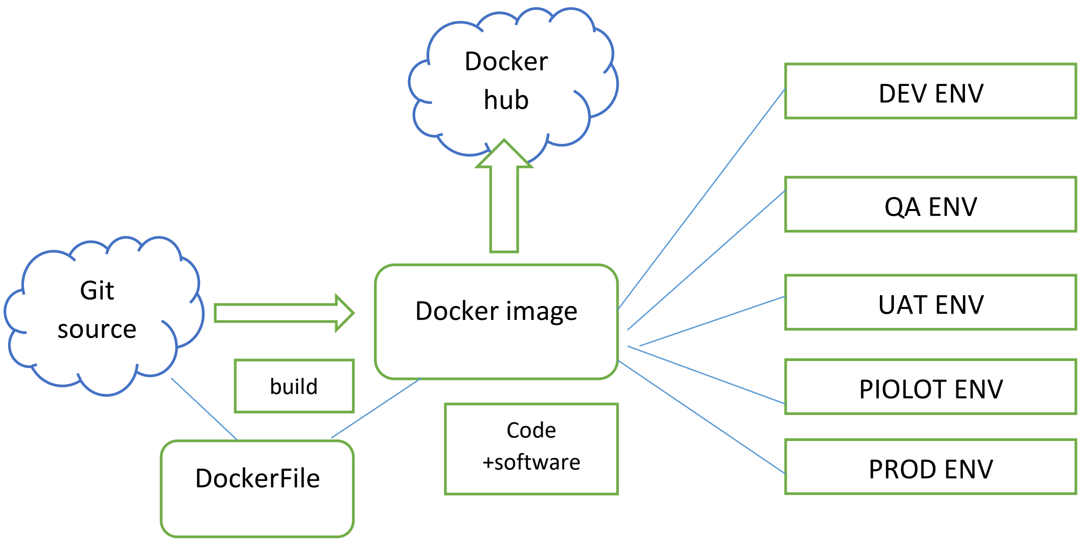
  $ docker rmi -f <image-id>

- # Stop Docker container

  $ docker stop <container-id>

- # Delete all stopped containers and un-used images and un-used networks

  $ docker system prune –a



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

WORKDIR

EXPOSE

VOLUME

=============

- FROM

=============

-> FROM keyword is used represent base image to create our our image

-> On Top of base image our image will be created

Syntax:

FROM java:jdk-1.8

FROM tomcat:9.5

FROM mysql:6.8

FROM python:3.3

=============

- **MAINTAINER**

=============

-> MAINTAINER keyword is used to specify Dockerfile author information

Syntax:

MAINTAINER Ashok <ashok.b@oracle.com>

=======

- COPY

=======

-> COPY command is used to copy the files from source to destination while creating docker image

Syntax:

COPY <source-location> <destination-location>

Ex:

COPY target/sbi-app.war   /app/tomcat/webapps/sbi-app.war

=======

- **ADD**

=======

-> ADD command is also used to copy files from source to destination while creating docker image

Syntax: ADD <source-location> <destination-location>

ADD <url> <destination-location>

Ex: ADD <URL>   /app/tomcat/webapps/sbi-app.war

Q) What is the difference between COPY and ADD commands ?

-> Using COPY command we can just copy the files from one path to another path with in the machine

-> Using ADD command we can copy files from one path to another path and it supports source location as URL also.

=======

- **RUN**

=======

-> RUN instructions will execute while creating the image

-> Using RUN we can give instructions to docker to execute commands

-> We can write multiple RUN instructions, docker will process all the RUN instructions from top to bottom

Example

RUN yum install maven

RUN yum install git

RUN git clone repo-url

RUN mvn clean package

=======

- **CMD**

=======

-> CMD instructions will execute while creating the container

-> Using CMD command we can run our application package file jar / war file

Example

CMD sudo start tomcat

Note: If we write multiple CMD instructions also docker will process only last CMD instruction. There is no use of writing multiple CMD instructions in one Dockerfile.

Q) What is the difference between RUN and CMD in Dockerfile ?

-> RUN is used to execute instructions while creating image

-> CMD is used to execute instruction while creating Container

-> We can write multiple RUN instructions in Dockerfile, docker will process all those instructions one by one.

-> If we write multiple CMD instructions in Dockerfile, docker willl process only last CMD instruction.

Sample Dockerfile

```
FROM ubuntu

MAINTAINER Ashok<ashokit@gmail.com>

RUN echo "Hi, i am RUN-1"

RUN echo "Hi, i am RUN-2"

CMD echo "Hi, I am CMD-1"

RUN echo "Hi, i am RUN-3"

CMD echo "Hi, i am CMD-2"
```

-> Save the above content in docker file

filename : Dockerfile

- **# Command to create docker image using dockerfile**

  Syntax **: $ docker build -t <image-name> .**

  Ex : $ docker build -t myfirstimage .

- # Command to run docker image

  $ docker run myfirstimage

- # Command to login with dockerhub account

  $ docker login

Note: We need to enter our dockerhub account credentials correctly (it will ask only first time)

- # Command to tag our docker image

  $ docker tag <image-name> <tag-name>

  Ex: $ docker tag myfirstimage ashokit/myfirstimage

- # command to push docker image to docker hub account

  $ docker push <tag-name>

Note: Delete all unused images and stopped containers

$ docker system prune -a

- # Pull the image from docker hub

  $ docker pull ashokit/myfirstimage

- # Run the image

  $ docker run ashokit/myfirstimage

Note: We can use customized name also for the dockerfile. When we change dockerfile name we need to pass filename as input for docker build command using -f option like below.

$ docker build -f <dockerfile-name> -t <image-name> .

============

- **ENTRYPOINT**

============

-> ENTRYPOINT instructions will execute while creating container

Syntax

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

ENTRYPOINT ["java" , "-jar" , "target/boot-app.jar"]

Q) What is the difference between CMD and ENTRYPOINT ?

-> We can override CMD instructions in runtime while creating container

-> We can't override ENTRYPOINT instructions

==========

- **WORKDIR**

==========

-> It is used to set working directory for an image / container

Ex: WORKDIR   /app/

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

======

- **ENV**

======

-> ENV is used to set Environment Variables

Ex: ENV <key> <value>

ENV   java   /etc/softwares/java

====

- ARG

====

-> It is used to remove hard coded values

-> Using ARG we can pass values in the runtime like below

```
Ex:     ARG branch

        RUN git clone -b $branch <repo-url>
```

```
$ docker build -t imageone --build-arg branch=master
```

=====

- USER

=====

-> We can set user for creating image / container

Note: After USER instruction all the remaining commands will execute with given user permissions

========

- **EXPOSE**

========

-> It is used to specify our container running PORT

Ex: EXPOSE 8080

Note: It is just like a documentation command to provide container running port number.

========

- **VOLUME**

========

-> VOLUME is used to specify docker container data storage location.

Note: Volumes are used for storage.

Dockerize Spring Boot Application

-> Spring Boot is one ready-made java based framework available in the market to develop java based applications quickly

-> Spring Boot is providing embedded server (internal server will be available, we no need to configure server for execution)

-> Spring Boot application will be packaged as jar file (mvn clean package goal will do that package)

Note: When we do maven package, project jar will be created in project target folder

-> To run spring boot applications we just need to run jar file like below

```
$ java -jar <boot-app-jar-file>
```

--------------------------------Dockerfile-----------------------------------

```
FROM openjdk:11

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

WORKDIR /usr/app/

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

Spring Boot App Git Repo URL : https://github.com/ashokitschool/spring-boot-docker-app.git

```
# install git client s/w

$ sudo yum install git

# Clone Git Repo

$ git clone https://github.com/ashokitschool/spring-boot-docker-app.git

# Navigating to project folder

$ cd spring-boot-docker-app
```

# install maven s/w

```
$ sudo yum install maven
```

# execute maven goals

```
$ mvn clean package
```

Note: After package got success, we can see project jar file in target folder.

# create docker image

```
$ docker build -t sb-app .
```

# run docker image with port mapping

```
$ docker run -p 8080:8080 sb-app
```

Note: Enable 8080 port number in EC2 VM security group

URL To Access Application :   http://ec2-vm-public-ip:8080/welcome/Ashok

How to run Docker container in Detached Mode

=> With below command our terminal will be blocked, we can't execute any other command. To execute other command we need to type CTRL+C then terminal will open for commands execution but our container gets stopped.

```
$ docker run -p 8080:8080 ashokit/sb-app
```

Note: To overcome above problem we can pass '-d' to run container in detached mode. When we execute below command it will run the container in detached mode and it will open terminal for commands execution.

```
$ docker run -d -p 8080:8080 ashokit/sb-app
```

=> Once above command is executed, we can see running containers using below command

```
$ docker ps
```

=> We can check logs of the container using below command

```
$ docker logs <container-id>
```

Dockerizing Java Web Application (Without SpringBoot)

-> Java web applictaions will be packaged as **war file**

-> WAR (Web Archive) contains application code

-> To run the war file we need webserver (Ex: Apache Tomcat)

-> We need to deploy war file in Tomcat Server for Execution

-> In Tomcat server we will have "webapps" folder for deployment

Note: To run normal java web applications we need "java & tomcat" as dependencies

```
-------------------------------------------------Dockerfile---------------------------------------------------
FROM tomcat:8.0.20-jre8
COPY target/01-maven-web-app.war   /usr/local/tomcat/webapps/maven-web-app.war
--------------------------------------------------------------------------------------------------------------
```

############### Java Web App Git Repo : https://github.com/ashokitschool/maven-web-app.git ############

```
$ git clone https://github.com/ashokitschool/maven-web-app.git
$ cd maven-web-app
$ mvn clean package
$ docker build -t maven-web-app .
$ docker images
$ docker run -d -p 8080:8080 maven-web-app
$ docker ps
$ docker logs <container-id>
```

URL To access The Application :   http://ec2-vm-public-ip:host-port/maven-web-app/

Note: In the above url "maven-web-app" is called as context path (name of the war file will become context path)

========================

Dockerize Python Application

========================

-> Python is a general purpose scripting language

-> Python programs will have .py as extension

-> Compilation is not required for python programs

```
------------------Dockerfile---------------------
FROM python:3.6
MAINTAINER Ashok Bollepalli "ashokitschool@gmail.com"
COPY . /app
WORKDIR /app
EXPOSE 5000
RUN pip install -r requirements.txt
ENTRYPOINT ["python", "app.py"]
```

Python Flask app Git hub Repo: https://github.com/ashokitschool/python-flask-docker-app.git

```
$ git clone https://github.com/ashokitschool/python-flask-docker-app.git

$ cd python-flask-docker-app

$ docker build -t python-flask-app .

$ docker images

$ docker run -d -p 5000:5000 python-flask-app

$ docker ps

$ docker logs <container-id>
```

- # Command to enter into docker container

  $ docker exec -it <container-id> /bin/bash

Note: Execute 'exit' command to come out from docker container vm.

Docker Network

- Network is all about communication
- Docker network is used to provide isolated network for Docker Containers
- In Docker we will have below **3 default networks**

1) none 2) host 3) bridge

- **In Docker we have below 5 network drivers**

1) Bridge ----> This is default network driver in Docker

2) Host

3) None

4) Overlay ----> Docker Swarm

5) Macvlan

- Bridge driver is recommended driver when we are running standalone container. It will assign one IP for container
- Host Driver is also used for standalone container. IP will not be assigned for container
- None means no network will be provided by our Docker containers

- Overlay network driver is used for Orchestration. Docker Swarm will use this Overlay network driver
- Macvlan driver will assign MAC address for a container. It makes our container as Physical.

❖ # display docker networks available

  $ docker network ls

❖ # Create docker network

  $ docker network create ashokit-network

❖ # Inspect network

  $ docker network inspect <network-name>

❖ # delete docker network

  $ docker network rm <network-id>

❖ # Run a container with given network

  $ docker run -d -p hport:cport --network ashokit-network <imagename>

Docker Compose

Monolith Application : One application which contains all the functionalities is called as Monlith App.

- If we make any small code change then we need to re-deploy entire application
- If we make any code change in one functionality there may be a impact on another functionality
- Maintenence will become very difficult when we go for Monolith Based Application

Note: To overcome the problems of Monlith we are using Microservices Architecture.

Microservices Application : Application functionality will be developed as micro services (rest apis)

- Every functionality will be developed as individual project (individual API)

ADMIN_API

REPORT_API

PAYMENT_API

CART_API

TRACKING_API

PRODUCT_API

CANCEL_API

- Every API should run in a seperate container
- Running Multiple containers manully for all the apis is difficult job

********** 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 easily setup & deploy multi container based applications
- We will give containers information to Docker Compose using YML file (docker-compose.yml)
- Docker Compose YML should have all the information related to containers creation

Docker Compose YML File

version:

services:

network:

volumes:

====================

- Docker Compose default file name is "docker-compose.yml"

❖ # Create Containers using Docker Compose

```
$ docker-compose up
```

❖ # Create Containers using Docker Compose with custom file name

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

❖ # Display Containers created by Docker Compose

```
$ docker-compose ps
```

❖ # Display docker compose images

```
$ docker-compose images
```

❖ # Stop & remove the containers created by docker compose

```
$ docker-compose down
```

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
  ```

Spring Boot with MySQL using Docker Compose

- Spring Boot App with MySQL DB Git repo URL : https://github.com/ashokitschool/spring-boot-mysql-docker-compose.git
- Below is the docker-compose file

```
---
version: "3"
services:
  application:
    image: spring-boot-mysql-app
    networks:
      - springboot-db-net
    ports:
      - "8080:8080"
    depends_on:
      - mysqldb
  mysqldb:
    image: mysql:5.7
    networks:
      - springboot-db-net
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=sbms
networks:
  - springboot-db-net:
...
```

Steps to Run Spring Boot application with MySQL DB using Docker Compose

- # Clone git repo url

  ```
  $ git clone https://github.com/ashokitschool/spring-boot-mysql-docker-compose.git
  ```

- # Get into project directory

  ```
  $ cd spring-boot-mysql-docker-compose
  ```

- # Build Maven Project

  ```
  $ mvn clean package
  ```

- # Create Docker Image (Image Name : spring-boot-mysql-app)

  ```
  $ docker build -t spring-boot-mysql-app .
  ```

- # Check docker image created or not

  ```
  $ docker images
  ```

- # Run containers using docker compose (docker-compose.yml available)

  ```
  $ docker-compose up -d
  ```

- # Check containers which are created

  ```
  $ docker-compose ps
  ```

- # Check logs of application contianer

  ```
  $ docker logs -f <container-name>
  ```

Note: Access the application in browser

URL : http://ec2-vm-public-ip:host-port/

Docker Volumes

- Applications we are executing using Docker Containers
- Docker containers are by default stateless
- Once container removed then we will loose the data that stored in the container
- In realtime we shouldn't loose the data even the container got removed

  For Example : Database container

- Application will store data in database, even if we delete application container or db container data should be available.
- To make sure data is available after the container is deleted we will use Docker Volumes concept

*************** Docker Volumes are used to store container data ****************

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- We have 3 types of volumes in Docker

1) Anonymous Volumes (without name)

2) Named Volumes (Will have a name) ----> Recommended

3) Bind Mounts (Storing on Host Machine)

Q) What is Dangling volume in Docker ?

-> The volumes which are created but not associated to any container are called as Dangling Volumes

- # Delete all dangling volumes

  ```
  $ docker volume rm $(docker volume ls -q -f dangling=true);
  ```

- # Create Docker volume

  ```
  $ docker volume create <vol-name>
  ```

- # Display all docker volumes

  ```
  $ docker volume ls
  ```

- # Inspect Docker Volume

  ```
  $ docker volume inspect <vol-name>
  ```

- # Delete docker volume

  ```
  $ docker volume rm <vol-name>
  ```

- #Delete all docker volumes

  ```
  $ docker system prune --volumes
  ```

------------------------------- **Docker Compose with Docker Named Volumne** ------------------------

```
version: "3"
services:
 application:
   image: springboot-app
   ports:
    - "8080:8080"
   networks:
    - springboot-db-net
   depends_on:
    - mysqldb
```

```
  mysqldb:
    image: mysql:5.7
    networks:
      - springboot-db-net
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=sbms
    volumes:
      - app_data:/var/lib/mysql
networks:
  springboot-db-net:

volumes:
   app_data:
```

Dockerizing Shopme app

- Connect to AWS Docker2 ec2 instance
- Check docker star if not start using

  $sudo service docker start

- $cd shopme-project
- $git pull
- $mvn clean -Dmaven.test.skip
- $cd..
- Check docker images
  Docker File

  ```
  FROM openjdk:15
  EXPOSE 9090
  COPY  /shopme-project/ShopmeWebParent/ShopmeBackEnd/target/ShopmeBackEnd-0.0.1-SNAPSHOT.jar /user/ShopmeBackEnd-0.0.1-SNAPSHOT.jar
  WORKDIR /user/
  COPY . ./
  ENTRYPOINT ["java","-jar","ShopmeBackEnd-0.0.1-SNAPSHOT.jar"]
  ```
- Check docker-compose.yml file
  version: "3"

```
services:
  application:
    image: shopme-backend-app
    ports:
      - "9090:9090"
    networks:
      - springboot-db-net
    depends_on:
      - mysqldb

  mysqldb:
    image: mysql:8.0.16
    networks:
      - springboot-db-net
    environment:
      - MYSQL_ROOT_PASSWORD=RRMrrm@#095
      - MYSQL_DATABASE=shopmedb

networks:
  springboot-db-net:
volumes:
  app_data:
```

- Run Command

  ```
  $docker-compose up –d
  ```

- Then application is running

Access application using

```
http://public-ip-aws:9090/ShopmeAdmin/
```

Note :

- To check application inside use command

  ```
  $ docker exec -it <container-id> /bin/bash
  ```

- To check database use

  ```
  $ docker exec -it ec2-user_mysqldb_1 /bin/bash
  root@661a388ee1c2:/#                          // will appear
  //type
  ```

  ```
  mysql -u root -p
  ```

Enter password: RRMrrm@#095

Mysql will be accessible