## Application Architecture

1) Frontend : User Interface (UI)

2) Backend : Business logic

3) Database : Storage

## Tech Stack of Application

Frontend : Angular 16v

Backend : Java 17v

Database : MySQL DB Server 8.5

Webserver : Tomcat 9.0

Note: If we want to run our application code, then we need to setup all required dependencies in the machine.

Note: dependencies nothing but the softwares which are required to run our application.

Ex: java 17 + Angular 16 + MySQL 8.5 + Tomcat server 9.0

Note: If we want to run same application in 100 machines then it is hectic task to setup dependencies and there is a chance of human mistakes.

=> To overcome above problem we will use Docker tool.

## What is Docker ?

=> Docker is a free & open source software.

=> Docker is used for containerization.

Note: Containerization means packaging application code + application dependencies as single unit for execution.

=> With the help of docker, we can run our application in any machine.

=> Docker will take care of dependencies installation required for application execution.

=> We can make our application portable using Docker.

Note: Docker is platform independent. We can use docker in windows, linux and mac also.

Docker Container = application code + application dependencies

## Docker Architecture

1) Dockerfile

2) Docker Image

3) Docker Registry

4) Docker Container

=> Dockerfile is used to specify where is app code and what dependencies are required for our application execution.

Note: Using dockerfile we will build docker image.

=> Docker image is a package which contains app code and app dependencies.

Docker Image = app code + app dependencies

=> Docker Registry is used to store docker images.

Note: When we run docker image then docker container will be created. Docker container is a linux virtual machine.

=> Inside Docker Container our application will be executed.

## Install Docker in Linux VM

Step-1 : Create EC2 VM (amazon linux) & connect with that vm using ssh client

Step-2 : Execute below commands

```
# Install Docker
sudo yum update -y
sudo yum install docker -y
sudo service docker start

# Add ec2-user user to docker group
sudo usermod -aG docker ec2-user

# Exit from terminal and Connect again
exit

# Verify Docker installation
docker -v
```

## Docker commands

docker images : To display docker images available in our system.

docker pull <image-id/name> : To download docker image from docker hub.

docker rmi <image-id/name> : To delete docker image.

docker run <image-id/name> : TO create/run docker container.

docker ps : To display running docker containers.

docker ps -a : To display running + stopped containers.

```
docker stop <container-id> : To stop running docker container.

docker start <container-id> : To start docker container which is in stopped state.

docker rm <container-id> : To delete docker container.

# delete stopped containers + unused images + build cache
docker system prune -a

docker build -t <tag-name> . : To build docker image

docker login : To login into docker hub account

docker push <img-name> : To push docker img into docker hub
```

## Running Real-world applications using docker images

```
docker pull ashokit/spring-boot-rest-api
docker run ashokit/spring-boot-rest-api
docker run -d ashokit/spring-boot-rest-api
docker ps
docker logs <container-id>
docker run -d -p host-port:container-port ashokit/spring-boot-rest-api

        Ex: docker run -d -p 9090:9090 ashokit/spring-boot-rest-api

########### Java App URL : http://public-ip:host-port/welcome/{name}


docker pull ashokit/python-flask-app
docker run -d ashokit/python-flask-app
docker run -d -p 5000:5000 ashokit/python-flask-app

########### Python App URL : http://public-ip:host-port/
```

Note: Here -d represents detached mode.
Note: Here -p represents port mapping. (host-port:container-port)

Note: host port and container port no need to be same.

Note: Host port number we need to enable in ec2-vm security group inbound rules to allow the traffic.

## Dockerfile

=> Dockerfile contains set of instructions to build docker image.

file name : Dockerfile

Note: We will keep Dockerfile inside project directory.

=> To write dockerfile we will use below keywords

1) FROM
2) MAINTAINER
3) RUN
4) CMD
5) COPY

6) ADD
7) WORKDIR
8) EXPOSE
9) ENTRYPOINT
10) USER

## FROM

=> It is used specify base image to create our docker image.

Ex:

```
FROM tomcat:9.0

FROM openjdk:17

FROM python:3.3

FROM node:19

FROM mysql:8.5
```

## MAINTAINER

=> To specify author of Dockerfile (who created/modifed Dockerfile)

Ex:

```
MAINTAINER Ashok<ashok.b@oracle.com>
```

Note: It is optional.

## RUN

=> RUN keyword is used to specify instructions (commands) which are required to execute at the time of docker image creation.

Ex:

```
RUN 'git clone <repo-url>'

RUN 'mvn clean package'
```

Note: We can specify multiple RUN instructions in Dockerfile and all those will execute in sequential manner.

## CMD

=> CMD keyword is used to specify instructions (commands) which are required to execute at the time of docker container creation.

Ex:

```
CMD "java -jar <jar-file-name>"

CMD "python app.py"

Note: If we write multiple CMD instructions in dockerfile, docker will execute only last CMD
instruction.

=====
COPY
=====

=> COPY instruction is used to copy the files from source to destination.

Note: It is used to copy application code from host machine to container machine.

                Source : HOST Machine

                Destination : Container machine

EX:

COPY target/app.jar       /usr/app/

COPY target/webapp.war /usr/app/

COPY app.py        /usr/app/


=====
ADD
=====

=> ADD instruction is used to copy the files from source to destination.


EX:

ADD target/app.jar /usr/app/

ADD <file-url>  /usr/app/


========
WORKDIR
========

=> WORKDIR instruction is used to set / change working directory in container machine.

Ex:

COPY target/app.jar /usr/app/

WORKDIR /usr/app/

CMD "java -jar app.jar"

========
EXPOSE
========

=> EXPOSE instruction is used to specify application is running on which PORT number.

Ex:

EXPOSE 8080
```

===========
ENTRYPOINT
===========

=> It is used to execute instruction when container is getting created.

Note: ENTRYPOINT is used as alternate for 'CMD' instructions.

```
CMD "java -jar app.jar"

ENTRYPOINT ["java", "jar", "app.jar"]
```

================================================
What is the diff between 'CMD' & 'ENTRYPOINT' ?
=================================================

CMD instructions we can override.

ENTRYPOINT instructions we can't override.

===========
USER
===========

=> It is used to set  user account to execute dockerfile commands

```
USER 'ashokit'
RUN echo 'hi'
```

===================================
Dockerizing SpringBoot application
===================================

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

=> Spring Boot is a java framework which is used to develop java based applications.

=> Spring Boot applications will be packaged as a jar file.

=> To run the jar file we will use below command

```
                Ex: java -jar app.jar
```

Note: When we run springboot application jar file, internally springboot will use tomcat server as "embedded container" with default port number 8080.

================ Java SpringBoot App Dockerfile ===========

```
FROM openjdk:17

MAINTAINER "Ashok"

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

WORKDIR /usr/app/

EXPOSE 8080
```

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

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

1) Clone git repo

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

2) Go inside project directory and perform maven build

```
cd spring-boot-docker-app
mvn clean package
```

3) Create docker image

```
docker build -t ashokit/sb-app .

docker images
```

4) Create docker container

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

docker ps

docker logs <container-id>
```

5) Access application URL in browser

windows : http://localhost:8080/

Linux : http://public-ip:8080/

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

Dockerizing Java Web application (no springboot)

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

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

=> Normal java web apps will be packaged as war file.

Note: war file will be created inside project target directory.

=> To execute that java web application we need to deploy that war file in tomcat server.

=> Inside tomcat server we will have "webapps" folder. It is called as deployment folder.

=> To run war file we need to keep war file inside tomcat/webapps folder.

================ Dockerfile for Java web application ===============

```
FROM tomcat:latest

MAINTAINER "Ashok<797979>"

EXPOSE 8080

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

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

1) Clone git repo

```
git clone https://github.com/ashokitschool/maven-web-app.git
```

2) Go inside project directory and perform maven build

```
cd maven-web-app
mvn clean package
```

3) Create docker image

```
docker build -t ashokit/maven-web-app .

docker images
```

4) Create docker container

```
docker run -d -p 8080:8080 ashokit/maven-web-app

docker ps

docker logs <container-id>
```

5) Access application URL in browser

```
        windows : http://localhost:8080/maven-web-app

        Linux : http://public-ip:8080/maven-web-app
```

## Dockerizing Python application

=> Python is a general purpose language

Note: It is also called as scripting language.

=> We don't need any build tool for python applications.

=> We can run python application code directley like below

```
        ex: python app.py
```

=> If we need any libraries for python (Ex: Flask) application development then we will mention them in "requirements.txt" file

Note: We will use "python pip" s/w to download libraries configured in requirements.txt file.

## Python Flask App Dockerfile

```
FROM python:3.6

COPY . /app/

WORKDIR /app/

EXPOSE 5000

RUN pip install -r requirements.txt

ENTRYPOINT ["python", "app.py"]
```

1) Clone git repo

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

2) Go inside project directory and Create docker image

```
cd python-flask-docker-app

docker build -t ashokit/py-app .

docker images
```

3) Create docker container

```
docker run -d -p 5000:5000 ashokit/py-app

docker ps

docker logs <container-id>
```

4) Access application URL in browser

        windows : http://localhost:5000/

        Linux : http://public-ip:5000/

## How to access docker container

```
# display docker containers which are in running mode
docker ps

# go inside docker container from linux host
docker exec -it <container-id> /bin/bash

# go inside docker container from windows host
docker exec -it <container-id> sh
```

## Assignments for today

1) Setup Jenkins Server as Docker Container

2) Setup MySQL DB as Docker Container

3) Dockerizing Angular & React application