**Docker (package once deploy/run anywhere)**

Docker is lightweight containerization technology which uses Union File System

Docker Server or Docker Daemon does majority of work in Docker

Docker Daemon runs on Docker Host

Not mandatory to do docker pull before docker run

Docker ps –a (list all containers running or stopped both)

Docker States: Running, paused, exited

A container will not lose data when it is stopped

A container will lose data when it is removed or deleted

1. Containerization

- Can work very fast

- Dev Test Stage

- Docker create a Docker Container using Docker Image once we dockerize.

- Docker launches Docker Conatiner using Virtualization Platform

- These docker containers can be deployed across environments as required

- Each container can hold a Application, DB Server, Web Server

2. Kubernetes

- Scale up and Scale down the containers

- Load Balancing

**Docker Architecture**

1. Registry
2. Docker Host + Docker Engine
3. Docker Client

docker pull <image>

docker run <image>

docker build <image>

yum install docker

docker --version

service docker start (starts docker daemon)

docker info

docker run hello-world (runs image from remote repo)

docker run --it ubuntu bash (runs a ubuntu bash)

docker images (show all images)

docker pull mysql:5.7 (pull from repo)

docker run --help (for help)

docker run --idtp 80:80 nginx (start nginx container)

docker images (show all images)

docker ps (process status of container)

docker ps --a (process status of all container)

local machine -> (connects)AWS box -> Container

docker ps --a

docker rmi <containerid> (delete container)

docker rmi <containerid1> <containerid2> (delete containers)

docker start <containerid> (start container)

docker stop <containerid> (stop container)

docker images

docker run --idtp 80:80 nginx (start nginx container)

docker run --dit --name=container\_nginx

docker start <containername> (start container)

docker stop <containername> (stop container)

docker run --idtp –name=myciontainer2 –p 8080:80 nginx (start nginx container)

docker ps

docker restart <containername> (restart docker)

docker pause <containername> (restart docker)

docker unpause <containername> (restart docker)

docker info

docker images

docker pull ubuntu

docker run –dit Ubuntu

docker exec –it <container-id> bash (to enter docker container)

apt-get update (get latest packages)

apt-get install apache2

service apache2 status

service apache2 start

service apache2 status

mkdir demo

ls

exit

docker ps

docker commit <container-id> mywebserver (store data in image webserver)

docker images

docker run –dit –name myweb mywebserver (launch container)

docker ps

docker exec –it myweb bash (to enter docker container)

service apache2 status

service apache2 start

service apache2 status

**Docker Layers and Overlay Storage**

docker history <image name>

**Run MySQL**

**(Launch MySQL container in detach mode)**

docker run –dit –p 6666:3306 –name=pushkar-mysql --env=”MYSQL\_ROOT\_PASSWORD=test1234” –env=”MYSQL\_DATABASE=emp” mysql

docker exec --it pushkar-mysql bash

mysql –u root –p <password>

show databases;

use emp;

exit;

**Dockerfile (to build your own image)**

Pull exiting images from Docker Hub & used those docker images to run container

exit;

Docker builds images from Dockerfile

Dockerfile contains commands to assemble the image

Docker CLI builds and executes Dockerfile which gives it to Docker Engine

Docker Engine creates image and Docker runs containers from images

FROM FROM java:8 or FROM ubuntu

COPY COPY app.jar /server/deploy

ENV ENV JAVA\_HOME=/opt/java

RUN RUN yum install apache2

EXPOSE EXPOSE 8080

CMD CMD python3 –version

ADD target/ couponservice-0.0.1-SNAPSHOT.jar couponservice-0.0.1-SNAPSHOT.jar

ENTRYPOINT [“java”,”-jar”,”couponservice-0.0.1-SNAPSHOT.jar”]

**Create first Dockerfile (to build your own image)**

Dockerfile

FROM centos

RUN yum install –y httpd

ADD index.html /var/www/html

CMD apachectl –D FOREGROUND

EXPOSE 80

MAINTAINER pushkar

ENV myenv myval

(Run dockerfile)

docker build –t my-webserver .

docker images (my-webserver created)

**Docker Layers**

docker images

docker history my-webserver

(Adding one more env variable to existing Dockerfile)

FROM centos

RUN yum install –y httpd

ADD index.html /var/www/html

CMD apachectl –D FOREGROUND

EXPOSE 80

MAINTAINER pushkar

ENV myenv myval

ENV myenv2 myval2

(Run dockerfile again Need not execute all commands again only updated)

docker build –t my-webserver .

docker images (my-webserver created)

(push image to central repo)

docker tag my-webserver pushkar/my-webserver

docker images

docker login

docker push pushkar/my-webserver

Validate from Docker Hub

**Dockerize Microservices**

Visual Studio Code (Add docker extension)

Run docker images

View docker images from docker hub

Push docker images to Docker Hub

Launch docker files from VSC

**Launch MySQL Container**

docker run --dp 6666:3306 --name=”docker-mysql” --env=”MYSQL\_ROOT\_PASSWORD=test1234” --env=”MYSQL\_DATABASE=mydb” mysql

docker ps

(open bash on container)

docker exec --it docker-mysql bash

mysql –u root –p

show databases;

use mydb;

docker exec –I docker-mysql mysql –uroot –ptest1234 mydb<tables.sql

**Dockerize couponService (in Visual Studio Code)**

Dockerfile

FROM java:8

ADD target/couponService-0.0.1-SNAPSHOT.jar couponService-0.0.1-SNAPSHOT.jar

ENTRYPOINT [“java”,”-jar”,”couponservice-0.0.1-SNAPSHOT.jar”]

**Dockerize productService (in Visual Studio Code)**

Dockerfile

FROM java:8

ADD target/productservice-0.0.1-SNAPSHOT.jar productservice-0.0.1-SNAPSHOT.jar

ENTRYPOINT [“java”,”-jar”,”productservice-0.0.1-SNAPSHOT.jar”]

**Build docker images for Microservices**

**CouponService : application.properties**

spring.datasource.url=jdbc:mysql://docker-mysql:3306/mydb

spring.datasource.username=root

spring.datasource.password=test1234

server.port=9091

**ProductService : application.properties**

spring.datasource.url=jdbc:mysql://docker-mysql:3306/mydb

spring.datasource.username=root

spring.datasource.password=test1234

server.port=9090

#couponService.url=http://localhost:9091/couponapi/coupons/

couponService.url=http://coupon-app:9091/couponapi/coupons/

mvn clean package –DskipTests

mvn clean package –DskipTests

**docker build –f Dockerfile –t product-app .**

**docker build –f Dockerfile –t coupon-app .**

**Launch Microservice container**

docker run –t –name=coupon-app –link docker-mysql:mysql --p 10555:9091 coupon\_app

docker run –t –name=product-app –link docker-mysql:mysql –link coupon-app:coupon-app --p 10666:9090 product\_app

**Test**

POST

{

“code”:”SUPERSALE”,

“discount”:”10”,

“expDate”:”12/12/2030”

}

<http://localhost:10555/couponapi/coupons>

{

“id”:1,

“code”:”SUPERSALE”,

“discount”:”10”,

“expDate”:”12/12/2030”

}

POST:

{

“name”:”MAC”,

“description”:”Its Cool”,

“price”:2000,

“couponCode”:”SUPERSALE”

}

<http://localhost:10666/productapi/products>

{

“id”:1,

“name”:”MAC”,

“description”:”Its Cool”,

“price”:1900.000,

“couponCode”:”SUPERSALE”

}

**Push images to docker hub**

docker tag coupon\_app pushkar/couponservice

docker tag product\_app pushkar/productservice

docker images

docker push pushkar/couponservice

docker push pushkar/productervice

Go to Docker hub ad validate for images

**Docker prune (remove all non running/stopped containers)**

docker container prune

**Docker prune (remove all running and non running containers)**

docker container prune –a

docker ps (view containers)

**Docker prune (remove all non running/stopped images)**

docker image prune

**Docker prune (remove all dangling images)**

docker image prune –a

docker images (view images)

**Docker remove all images**

docker rmi $(docker images –a -d)

**Docker remove all images**

docker image prune -a

docker images

**Docker remove all images and all containers, volumes**

docker system prune

docker volume prune

docker volume ls

**Docker Compose**

Docker Compose: Run one or more containers required for our microservice application

Launch multiple containers at a time by creating a docker-compose.yml file

Using docker-compose we will create one single dockerfile within that file each container will be defined as a service

**docker-compose up will run all containers / multiple containers**

Each service will depend on another service

Docker-compose options

docker-compose up

docker-compose stop

docker-compose restart

docker-compose build

docker-compose scale

docker-compose stop

docker-compose kill

docker-compose logs (check output of each container)

docker-compose ps (all running containers)

docker-compose help

**Yaml(Yaml Ain’t Markup Language)(Used in dockercompose, kubernetes, ansible)**

#Comments in Yml

#YAML is human readable

#YAML is superset of JSON

#Simple Types

#1. List 2.Objects 3. Dictionary/Map

customerName: Pushkar

orderId: 123

date: 23/07/2020

shipped: false

moreData: null

products:

* MacBook
* IPhone
* Desktop

address:

street: Spring

city: Austin

**docker-compose.yml for MYSQL, couponService and productservice**

version: '3'

services:

product-app:

container\_name: product-app

image: pushkar/productservice

restart: on-failure

ports:

- '10666:9090'

environment:

WAIT\_HOSTS: 'mysql:3306'

depends\_on:

- docker-mysql

- coupon-app

coupon-app:

container\_name: coupon-app

image: pushkar/couponservice

restart: on-failure

ports:

- '10555:9091'

environment:

WAIT\_HOSTS: 'mysql:3306'

depends\_on:

- docker-mysql

docker-mysql:

container\_name: docker-mysql

environment:

MYSQL\_DATABASE: mydb

MYSQL\_ROOT\_HOST: '%'

MYSQL\_ROOT\_PASSWORD: test1234

heathcheck:

internal: 4s

retries: 5

test: >-

/usr/bin/mysql --user=root --password=test1234 --execute "SHOW

DATABASES"

timeout: 20s

image: mysql

ports:

- '6666:3306'

restart: always

volumes:

- ./sql/docker-entrypoint-initdb.