Docker Notes by Praveen Challa

MONOLITHIC:- if an application consists of N number of services, if all these services are included in one server then it will be called as Monolithic Architecture. every monolithic Architecture has only one database for all the services

MICRO SERVICE:- if an application contains n number of services. if every service has its own individual servers then it is called microservices. every microservice architecture has its own database for every service

VIRTUALIZATION:

It is used to create a virtual machines inside on our machine. in that virtual machines we can host guest OS in our machine

by using this guest OS we can run multiple application on same machine.

------->Hypervisor is used to create the virtualization.

DRAWBACKS

--> it is old method

--> if we use multiple guest OS then the performance of the system gets slow

CONTAINERIZATION: it is used to pack the application along with its dependencies to run the application.

ex: IMAGE=WAR(appliaction)+JAVA+TOMCAT+MYSQL

JAVA,TOMCAT are dependicies.

CONTAINER: container is nothing but a virtual machine which does not have any OS-----> Docker is used to create these containers

------>if we run image on any server/machine then container will gets created and application will run

**DOCKER:

-->it is an open source centralized platform designed to create, deploy and run applications.

-->docker is written in Go language

-->docker uses containers on host OS 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 OS

-->we can install docker on any OS but the docker engine runs natively on linux distribution.

-->docker performs OS level virtualization also know as Conyainerization

**Docker has 4 components

1)Docker client: docker client is nothing but where we perform the commands

2)Docker Host: docker host is nothing but where we install the docker like ec2

(machine where you installed the docker engine.)

3)Docker daemon: daemon contains containers,images, volumes and Networks

4)Registry: like docker hub here we can store the images and share the images with others

(scalable open-source storage and distribution system for docker images)

POINTS TO BE FOLLOWED:

-->You cant use docker directly, you need to start/restart first

-->you need a base image for creating a container.

-->you cant enter directly to container, you need to start first

-->if you run a image, by default one conatiner will create.

DOCKER--->CONTAINER---->APPLICATION

launch the server and follow me.

yum install docker -y (used to install docker)

docker version (used to check the docker version)

systemctl status docker (uesd to check status of docker)

systemctl start docker (used to start docker)

docker images (used to check the list of images)

docker run ubuntu (for creating container we have to some base image like ubuntu, tomcat, java etc)

docker ps -a (to check the list of containers) docker run -it --name shanku ubuntu (used to create a container)

here

it-->interactive terminal which creates a shell inside container --->shell is used to perform commands

shanku-->this is name of the container

ubuntu-->this is image name.

exit (used to exit from container and goto local machine)

docker attach containername/containerid (used to go inside container but here we cant go dirctly first we have to start the container)

docker start containername (used to start the container)

ctrl+p+q (used to come to local machine without exiting the container)

docker rm containername/containerid (to remove container)-->here we can't remove running containers

docker stop ${docker ps -a -q} (used to stop all containers at a time)

docker rm ${docker ps -a -q} (used to remove all containers at a time)

docker rmi imagename (used to remove the image)-->here to remove images we have to remove all running containers which are running on that image)

**Difference between Docker run and Docker Pull

DOCKER RUN

-->when we use docker run command for creating image here one default container will also gets created

-->docker run--->docker daemon---->images--if there-->creates container

-->docker run--->docker daemon---->images--if not there-->registry---->creates container

-->when we use docker pull command the image only gets created

-->docker pull------->Registry--->creates container

docker pull ubuntu (used to create a image by using pull)

docker ps (used to check only running containers)

docker stop containername containername (used to stop multiple containers at a time)

docker ps -f "status=exited" (used to get only exited containers)

docker container ls -n 2 (used to get 2 latest running containers)

docker container ls -a -n 2 (used to get 2 latest containers)

docker container ls --latest (used to get the latest container)

docker container ls -a -q (used to get all container ids)

docker container ls -a -s (used to get running containers along with used space) docker container stats (used to monitor containers like to know about mem usage, cpu, limit, id.)

docker kill containername (used to stop the container it takes less time compare to docker stop command)

docker pull nginx (used to create web server image)

docker run -it --name cname -d -p 8081:80 nginx (used to create container with nginx server image)

here -d is detached mode nothing but application should work foreground as well as background)

-p is used to pubblish port number

8081-->host port

80--> container port (it is image default port number)

docker run -it --name cname -d -p 8082:80 httpd (used to create container with httpd server image)

here wecant go inside container but we can access container through publicip:portnumber

docker exec -it containername bash (used to go inside container)

docker exec cname command (used to perform commands without goinginto container)

ex: docker exec cont1 ls

docker container prune (used to delete exited containers)

creating image by container

docker commit containername imagename (used to create image from container)

-->it is used for backup purpose in realtime

**Docker file:

-->it is basically a text file which contains some set of instructions.

-->Automation of Docker image creation

-->Always D is capital letters on docker file

-->start components also be capital letter.

How it Works:

-->first you need to create a dockerfile ---->build it-->create a container using the image.

Here are some essential commands you should familiarize yourself with before using Docker.

FROM: it will takes the base image (ubuntu, httpd, nginx, tomcat)

LABEL: Author of a file

COPY: used to copy the files from local to container

ADD: used to copy the files local to container and also it will download the files from internet and send to container

EXPOSE: used to publish port number (8081,8082)

WORKDIR: it will creates a folder and we will directly goes into the path

RUN: used to execute commands while we build the image

CMD: used to execute the commands while we run the image(creating the container)

ENTRYPOINT: used to execute the commands while we run the image. entrypoint will have high priority than cmd

ENV: used to store the values-->it is not possible to over write the values

ARG: used to store the values -->it is possible to over write the values-->we cannot access these values inside the container

-->vim Dockerfile (used to create a docker file)

Docker file for creating a file aws.txt, devops folder, copy file, download file and to create workdirectory as myapp

```
FROM ubuntu

LABEL name shanku

EXPOSE 8081

RUN touch aws.txt

RUN mkdir devops

WORKDIR /myapp

COPY sourcefile destpath

ADD url path
```

here WORKDIR is important as execution start from top if we give workdir after label, file and folder gets created in that directory and if we give WORKDIR at the end states file and folder creates in a current directory then container creates in that folder

-->if we use ENTRYPOINT we can give package names while creating a container also it will creates

like docker run image2 httpd tree but

if we give multiple ENTRYPOINTS in Docker file then the latest ENTRYPOINT only will gets executed.

-->if we give one CMD and one ENTRYPOINT -->ENTRYPOINT only gets executed beacause ENTRYPOINT have more priority than CMD docker build -t image1 . (used to build the docker file)

here

-t is label

image1 is image name

. indicates the path of the file

docker run -it --name cont1 image1 (used to create a container by using image1) here we can create multiple containers with the same image name by giving tag like

docker run -it --name cont2 image1:1

cont2 is container name and image1:1 is iamge name.

----

creating a Docker file for storing values

```
FROM centos:centos7

ENV name=shanku

RUN echo "my name is $name"

ARG course=devops

RUN echo "my course is $course"
```

docker build -t image1 . --build-arg course=aws

ARG values can override. while ENV are not

----

create a folder myapp

create a file index.html in myapp folder

-->write code in index.html file

Docker file to bulid a image

```
vim Dockerfile

FROM ubuntu

RUN apt update -y

RUN apt install apache2 -y

ADD index.html /var/www/html/

CMD ["/user/sbin/apachectl", "-D", "FOREGROUND"]
```

Fork the repository digital_marketing in github

install git in our server

-->git clone url (to clone the code)-->cd digitalmarketing-->cd 2128_tween_agency/-->vim Dockerfile

```
F["ROM ubuntu

RUN apt update -y

RUN apt install apache2 -y

COPY . /var/www/html/

CMD ["/user/sbin/apachectl", "-D", "FOREGROUND"]
```

----

for static site

--in root

```
git clone url

cd stat..

cd html/
```

writing Dockerfile for centos image and httpd server

```
vim Dockerfile
```

```
FROM centos:centos7

RUN yum install httpd -y

COPY ./var/www/html/

CMD ["/user/sbin/httpd", "-D", "FOREGROUND"]
```

```
docker build -t image3 . (used to build image3)
```

```
docker run -itd --name cont3 -p 8083:80 image3 (used to create container
```

----

in root git clone url(job_portal)

```
cd job_portal

cd 2134..

vim Dockerfile
```

```
FROM ubuntu

RUN apt update -y

RUN apt install nginx -y

COPY . /var/www/html/

CMD ["nginx", "-g", "daemon off;"]
```

docker build -t image4 .

docker run -itd --name cont4 -p 8084:80 image4

----

in root git clone url(tic_tac..)

cd tic./

writing a Dockerfile for installing tic_tac_toe game using nginx image

vim Dockerfile

FROM nginx

COPY . /usr/share/nginx/html/

docker build image5 .

docker run -itd --name cont5 -p 8085:80 image5

----

in root git clone url(clean site)

cd clean_site

cd clean_work

writing a docker file for httpd image

vim Dockerfile

```
FROM httpd
COPY . /usr/local/apache2/htdocs/
```

```
docker build -t image6 .
docker run -itd --name cont6 -p 8086:80 image6
```

ABOVE TASKS

UBUNTU OS---> APACHE2

CENTOS ---> HTTPD

UBUNTU --->NGINX

NGINX

HTTPD

----

deploying application server using dockerfile

in root git clone url(one)

```
cd one
git checkout master
```

```
yum install java-1.8.0-openjdk -y
```

```
yum install maven -y
```

mvn clean package

vim Dockerfile

```
FROM tomcat:8.0.20-jre8

COPY tomcat-users.xml /usr/local/tomcat/conf/

COPY target/*.war /usr/local/tomcat/webapps/
```

save and quit

docker build -t image7

docker run -itd --name cont7 -p 8087:8080 image7

copypaste ipadress:8080-->managerapps--> we cant access

dlcdn-->tomcat-->9-->copy link adress-->server

one--wget url

one--tar -zxvf apache...

one--cd apache-tomcat-9.0.75/conf/

conf--cp tomcat-users.xml ../../

conf--cd ../../

one--rm -rf a*

one--vim tomcat-users.xml-->copy 3 lines and paste at the end and make changes.

----

yum install docker -y && systemctl restart docker

docker version

docker run -it --name cont1 -v /volname ubuntu (used to create volume along with container)

cd volname-->touch file{1..3}-->ctrlpq

docker inspect contname (used to check about volume info)

docker run -it --name cont2 --privileged=true --volumes-from cont1 ubuntu

here (create one file) touch aws-->(quit from cont2) ctrlpq-->(goto contl) docker attach cont1-->(check for file aws) ll.

-->we can delete files and folders inside the volume but we cant delete the volume

docker commit cont2 image1(used to create image from cont2)

docker run -it --name cont4 image1 (used to create container from image1)

-->we cant share/delete volumes in existing containers

-->we can access volumes in stoped containers also

-->we can create multiple volumes in one container docker stop ${docker ps -a -q} (used to stop all containers)

docker container prune (used to delete stoped containers)

docker volume inspect volumeid (used to inspect docker volume)

cd /var/lib/docker/-->ll-->cd volumes-->cd volID-->ll-->cd _data-->ll (used to access the volume in deleted container)

creating volume through docker file

vim dockerfile

FROM ubuntu

VOLUME ["/shanku-volume"]

VOLUME ["/challa-volume"]

docker built -t image2 . (used to build docker image)

docker run -it --name cont1 image2 (used to create container from image2)

docker run -it --name cont2 -v /home/ec2-user:/adc-volume --privileged=true ubuntu (used to share volume from host to container)

or

docker run -it --nmae cont3 -v "${pwd}":/abc-volume ubuntu

docker volume ls (used to know all created list of volumes) docker volume create aws (used to create aws volume outside of container) and docker volume rm aws (used to remove the volume)

docker volume prune (used to delete unused volumes)

Now we are attaching one volume to container

docker volume create Azure-->docker volume inspect Azure-->copy mountpoint-->cd path-->here create some files

docker run -it --name cont5 --mount source=Azure,destination=/devops ubuntu (used to attach existing volume to the new container)

DOCKER NETWORKS

DOCKER network:

it allows you to attach your container into many networks, it is used to isolate

the containers.

COMMANDS RELATED TO NETWORK:

if we want to know the connection between two containers

docker attach cont1-->apt update -y-->apt install iputils-ping -y-->ping cont2ipaddress-->if we get logs then we have connection.

docker network create net1 (used to create a docker network)

docker network ls (used to check list of networks)

docker run -it --name cont6 --network net1 ubuntu (used to create a container with net1 network)

docker network connect bridge cont6 (used to connect bridge network to container)

docker network disconnect bridge cont6 (used to disconnect bridge network to container)

docker network rm nrtworkname (used to delete the nrtwork)

docker network prune (used to delete all unsed networks)

DOCKER HUB

-->Docker Hub is a cloud-based registry service provided by Docker. It serves as a central repository for Docker images, allowing users to store, share, and distribute their container images. Docker Hub provides a convenient platform for developers and organizations to collaborate and access a wide range of pre-built Docker images.

yum install docker -y

systemctl restart docker

```
yum install git -y

git clone url(tic tac toe)

vim Dockerfile

FROM nginx
COPY . /usr/share/nginx/html

save and quit

docker build -t image1 .

docker run -itd --name cont1 -p 8081:80 image1
```

to access the container publicip:8081

----

hub.docker.com-->username-->email-->password-->tick-->tick-->signup-->personal

repositories-->create repository-->Name-->public/private-->create

before pushing the image to dockerhub we need to tag the image

```
docker tag image1 accname/reponame (used to tag a image)
```

docker login-->username-->password-->if login succedd-->docker push accname/reponame-->done

install docker -->docker pull repourl-->docker run -itd --name cont1 -p 8081:80 repourl-->done (used to create container from dockerhub image)

------------------for automating----------

1)setup jenkins in server-->git clone url(digital marketing)-->cd digital marketing-->cd 2128_tween_agency/-->vim Dockerfile-->

```
FROM nginx

COPY . /usr/share/nginx/html/
```

push dockerfile to github

2)-->git add *-->git commit -m "Docker file is ready"-->git push-->username-->password(token)

3)jenkins-->create a pipeline job-->save-->Build now

```
pipeline {

 agent any

 stages {

  stage ("code"){

    steps {

       url(pipeline syntax-->sample step(git)-->repo url-->branch(fim)-->genererate pipeline script-->url

         }

       }

       stage ("build") {

         steps {

           sh 'docker build -t image2 2128_tween_agency'
        }

       }

       stage ("Deploy"){

         steps {

         sh 'docker run -itd --name cont2 -p 8082:80 image2"

         }

        }

       }

     }
```

here we can do full automatic by using build with parameters concept.

-->we can check running containers and images from jenkins dashboard also

----

DOCKER SWARM

-->Docker swarm is an orchestration service within docker that allows us to manage and handle multiple containers at the same time.

-->It is a group of servers that runs the docker application.

-->It is used to manage the containers on multiple servers.

-->This can be implemented by the cluster.

-->The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster is called swarm worker

lauch 3 instance and install docker in all 3

yum install docker -y && systemctl restart docker (used and install docker and restart docker)-->perform the command in all 3 servers in these 3 server one is manager server and remaining are worker servers

in the manager server install docker swarm

docker swarm init --advertise-addr ipaddress(private) (used to initialize the docker swarm)-->This will generates a token if we perform the token in remaining server those will join like a worker

docker node ls (used to get list of docker nodes)

docker service create --name myapp(service name) --publish 8081:80 nginx (used to create a service)

docker service ls (used to get list of services)

docker service create --name servicename --publish 8082:80 --replicas 5 httpd (used to create 5 replicas)

docker service rm servicename servicename (used to remove the service)

docker swarm leave (used to leave the node server from swarm)

docker node rm nodename(used to remove the node in manager server)

docker swarm join-token worker (used to get token in manager server to join worker server)

docker swarm join-token manager(used to get token in manager server to join manager server)

CREATING A SERVICE USING LOCAL IMAGE

-->install git in server

-->git clone url(digital marketing)

-->write a docker file

-->build the docker file

-->docker tag imagename accname/reponame (used to tag a image)

-->push the image to dockerhub (docker login-->give username and password)

-->docker push imagename (used to push the image to dockerhub)

**NOTE:- The requirement is to develop a service that includes an image. In order to distribute the service effectively, we need to upload the image to DockerHub, which serves as a centralized repository. If we create the service without storing the image locally or in a central repository, all containers will be generated on the manager node exclusively.

docker service create --name website --publish 808_:80 --replicas 5 imagename (used to create a service using image)

docker service scale website=15 (used to scale up website service to 15 containers)

docker service scale website=3 (used to scale down website service to 3 containers)

docker service update --image newimage servicename (used to update the image to existing service)

docker service rollback servicename (used to rollback to previous image/version)

DOCKER COMPOSE

-->It is a tool used to build, run and ship the multiple containers for application.

-->It is used to create multiple containers in a single host.

-->It used YAML file to manage multi containers as a single service.

-->The Compose file provides a way to document and configure all of the application’s service dependencies (databases, queues, caches, web service APIs, etc).

install docker in server

vim docker-compose.yml/vim docker-compose.yaml (used to write docker compose file)

writeing a docker compose file for creating container with service name myapp and work on image nginx, also swiggy service work on httpd image.

```
version: "3"
services:
 myapp:
   image: nginx
   ports:
    - "8081:80"

 swiggy:
    image: httpd
    ports:
     - "8082:80"
```

docker-compose up -d (used to execute the compose file)

docker-compose down (used to delete the container)

docker-compose stop (used to stop the containers)

docker-compose images (used to get the list of images used in compose)

docker-compose logs (used to get list of logs)

ADDING VOLUMES AND NETWORKS TO THE CONTAINERS

```
version: "3"
services:
 myapp:
   image: nginx
   ports:
    - "8081:80"
   volumes:
    - "volume1"
   networks:
    - "network-1"

 swiggy:
    image: httpd
    ports:
     - "8082:80"
    volumes:
     - "volume2"
    networks:
```

```
    - "network-2"


networks:
  network-1:
    driver: bridge


  network-2:
    driver: bridge
```

NOTE: HERE WHENEVER WE ASSIGN THE NETWORK TO THE CONTAINER WE HAVE TO ASSIGN THE DRIVERS ALSO

DEPLOYING THE APPLICATION USING DOCKER COMPOSE

step-1: write the code in index.html file (vim index.html)

step-2: write the docker file for index.html file

step-3: build the image

step-4: write the docker-compose and execute

PROJECT

-->install git

-->git clone url(digital_marketing)-->write Docker file

-->git clone url(job_portal)-->write Docker file

-->write docker-compose file for both the services

```
vim docker-compose.yml
```

```
version: "3"
services:
 job_portal:
   build: path of dockerfile
   ports:
    - "8081:80"
   volumes:
    - "volume1"
   networks:
    - "network-1"

 digital:
    build: path of dockerfile -->ex: ./
    ports:
     - "8082:80"
    volumes:
     - "volume2"
    networks:
     - "network-2"

networks:
   network-1:

     driver: bridge


   network-2:

     driver: bridge
```

DOCKER STACK:

SWARM: SINGLE CONTAINER-MULTIPLE HOSTS

COMPOSE: MULTIPLE CONTAINERS-SINGLE HOSTS

STACK: MULTIPLE CONTAINERS-MULTIPLE HOSTS

-->Docker Stack is a command-line tool and configuration format used in Docker to deploy and manage multi-container applications. It is primarily designed for orchestrating and scaling services across a Docker swarm cluster, which is a group of Docker nodes working together.

-->A Docker stack is defined using a YAML file called a "Compose file" (typically named docker-compose.yml). This file specifies the services, networks, volumes, and other configurations required for the application. Each service represents a containerized component of the application, such as a web server, database, or message queue.

-->By using Docker Stack, you can deploy and manage multiple services as a single unit, enabling easy scaling, updating, and monitoring of the application. Some key features and concepts related to Docker

HERE TO PERFORM DOCKER STACK WE HAVE TO INSTALL DOCKER SWARM AND DOCKER COMPOSE IN OUR SERVER for docker stack also we write docker compose file

docker stack file for nginx image

vim docker-compose.yml

```
version: "3"
services:
 swiggy:
   image: nginx
   ports:
    - "8081:80"

  zomato:
   image: httpd
   ports:
    - "8082:80"
```

docker stack deploy --compose-file composefilename stackname (docker-compose.yml)--(used to execute the docker stack file

docker stack ls (used to get list of stacks)

docker stack ps stackname (used to get list of containers to that stack)

DOCKER PORTAINER/DOCKER GUI:

-->Docker Portainer is an open-source graphical user interface (GUI) management tool for Docker environments. It provides a user-friendly web interface that allows users to easily manage and monitor Docker containers, images, networks, and volumes. Portainer aims to simplify Docker container management tasks and make them more accessible to users without extensive command-line experience.

docker portainer is to be performed in docker stack only

-->9000 is portainer default port number

-->it is a container organizer, designed to make tasks easier, whether they are clustered or not.

-->abel to connect multiple clusters, access the containers, migrate stacks between clusters

-->it is not a testing environment mainly used for production routines in large companies.

-->Portainer consists of two elements, the Portainer Server and the Portainer Agent.

-->Both elements run as lightweight Docker containers on a Docker engine

Portainer commands to install:

Must have swarm mode and all ports enable with docker engine

```
curl -L https://downloads.portainer.io/ce2-16/portainer-agent-stack.yml -o portainer-
agent   stack.yml
```

```
docker stack deploy -c portainer-agent-stack.yml portainer
```

```
docker ps
```

-->public-ip of swamr master:9000

Docker directory data:

Docker containers often have a designated directory where data can be stored persistently. This directory is typically mapped from the host system to the container using a volume or bind mount. It allows the container to access and store data outside of its ephemeral filesystem, ensuring data persistence even when the container is restarted or replaced.

We use docker to run the images and create the containers. but what if the memory is full in instance. we have a add a another volume to the instance and mount it to the docker engine.

Lets see how we do this.

Uninstall the docker - yum remove docker -y

remove all the files - rm -rf /var/lib/docker/*

create a volume in same AZ & attach it to the instance

to check it is attached or not - fdisk -l

to format it - fdisk /dev/xvdf --> n p 1 enter enter w

set a path - vi /etc/fstab (/dev/xvsf1 /var/lib/docker/ ext4 defaults 0 0)

mount -a

install docker - yum install docker -y && systemctl restart docker

now you can see - ls /var/lib/docker

df -h
