Containers

Docker is a containerisation platform.

Containers are very lightweight because they don’t have their own operating system(containers have minimal operating system) n if they require any dependencies or system libraries then they will take it from host operating system.

To solve problem working with physical servers we started using virtual machine n again using virtual machine we were not using so many resources so to avoid that we started using containers but container are not secure compared to virtual machines

We can create containers on top of physical server or virtual machines but 2nd way is more preferred , maintaining physical servers is not easy so it’s good to go with virtual machines

When we try to take snapshots in virtual machines they it will take 2 or 3 gb storage but using containers we can create image in mbs

Docker file 🡪 docker build 🡪 docker image 🡪 docker run 🡪 docker container

Here we can face single time failure as if docker engine goes down then we can’t access containers n apps

Containers is standarlized unit of software that packages code n all dependencies so that application will run quickly n easily from one computing environment to another

Docker is a containerization platform that provides easy way to containerize applications.

docker image file n folders:: bin: binary files, sbin: system binaries, etc: configuration files, root: home directory for root user, var: var data, log files, lib: library files, usr: user related files

files n folders using from host operating system:: host system’s file, network stack, control groups, system calls, namespaces

registry: platform to store docker images

container vs virtual machines

resource utilization: containers take required resources from host operating, virtual machines are resource intensive, having full fleged operating system

portability: we can run containers anywhere with compatible host operating system, virtual machines we can run only where compatible hypervisor is there

security: containers are not secure compared to vms as they share kernel with host operating system, virtual machines are secure

docker daemon: It is a brain of docker, if we kill it then containers are not accessible, it listen to docker api requests n manages docker objects such as images, containers, volumes

docker registery: we use it to store docker images, docker hub is public registery

docker image: it is a read only template with instructions to create container

client(docker build(create docker images), docker run(create docker containers), docker pull(pull docker images from registries)) 🡪 docker host(docker daemon)image, container 🡪 registery(nginx)

docker client: it is primary way to communicate with docker daemon

dockerfile: it contains instructions to create docker image

**First practical**

Run yum update

Install docker

Check docker status

Sudo systemctl docker status

To check is ready to use or not

Docker run hello

Add Ubuntu user to docker gp

Sudo usermod add –aG docker Ubuntu

Take restart

Cat app.py

Dockerfile

Docker build –t userid/reponame:latest .

Docker run –it userid/reponame:latest

Docker hub 🡪 Docker login 🡪 docker push

Docker images

Django App deployment

First install python

Install django

Install django admin

Django admin startproject projectname it will create skeleton for app

In settings.py we can store templates , database details, ips, secret keys

In urls.py responsible for serving content

So these is just skeleton of application

So to create app run python manage.py startapp polls

Views.py 🡪 action code it will create html file

That html file we place in template folder from here content get served

Devops engineer get a task to containerize django application

To containerize application we need to start writing docker file

Work dir 🡪 here we we store source code so it should be at one place

Copy requirments.txt 🡪 python dependencies

Copy devops 🡪 copied source code

Using dependencies n source code we can form binary of an app

Install python

To download dependencies 🡪 pip install requirements.txt && \

cd devops

when someone runs docker run both entrypoint n cmd used as starting command

entrypoint 🡪 cannot change we can’t override value in docker image

cmd 🡪 we can change value it is configurable

**Second practical**

git clone <https://github.com/iam-veeramalla/Docker-Zero-to-Hero.git>

cd /home/ubuntu/Docker-Zero-to-Hero/examples/python-web-app

docker build .

docker images

docker run –it dockerimageid 🡪 not working

docker run -p 8000:8000 -it dockerimageid

<http://54.88.96.139:8000/demo/>

dockerfile in this we are writing steps to build docker image

but for running application we just need python runtime

to install python dependencies we use Ubuntu image but it will overload n image size will became big so to avoid that we introduced new concept that is multi stage docker

here suppose we divide our dockerfile into two parts

in first part we write FROM Ubuntu as Build

first part

FROM Ubuntu as build

Run python install 🡪 this binary we can directly use in second part to reduce size just the content required for runtime that we carry from here

Second part

From python (we can take image having just python runtime)

Copy --from build

CMD

First part will not be there in final image it’s only there in build image

Final stage will just have runtime + binary that we build in stage first + executable(CMD)

We can create countless stages in multistage docker build, there will be only one final stage which will be a minimalistic image

Distroless 🡪 very minimalistic image

By moving to multistage n distroless we are not only making sure that size is less but also it is secure n very less vulnerable to threats

Previously while using Ubuntu images we were exposed to vulnerabilities , people may face somekind of issues so we move to distroless images if we using python application we can can move it to distroless image it only have python runtime not even have basic packages like ls cp n after impleting distroless image we can say our application not exposed to ops related vulnerabilities

**Third practical**

Cd /home/ubuntu/Docker/Docker-Zero-to-Hero/examples/golang-multi-stage-docker-build

go run calculator.go

cd dockerfile-without-multistage/

docker build –t simplecalculator .

docker image | head -5

cd ..

vi Dockerfile

docker build -t withmutlistagedocker .

docker image

if container goes down then we can’t access files which were present on container so to overcome this issue bindmounts came into picture it will bind container dir with host dir

volume it provides offer better life cycle, logical partition, create volume on hots that we can mount bind to container

**Fourth practical**

docker volume ls

docker volume create sumati

docker volume inspect sumati

docker volume rm sumati

mount volume on container

cd /home/ubuntu/Docker-Zero-to-Hero/examples/first-docker-file

docker build -t volume .

docker images

docker volume create sumati

docker run -d --mount source=sumati,target=/app volumedemo:latest

docker inspect containerid

to delete volume first stop,delete container

Networking allows containers to talk with each other n with host system

Eth0 network is by default created

Whenever we create container the veth network created by default to talk with host which is called as bridge networking

Host n container are in diff subnets so it’s connected using a bridge veth

Host networking 🡪 whenever we create container it will bind it’s ip with eth0 of host, container will use host networking which is not secure, whoever has access to host they can access containers also which is not secure

Overlay networking 🡪 if we have multiple hosts n we have to make cluster connect all this then we can use

If we use default networking while connect two containers to host like we connecting to veth that id docker0 n connecting to docker but there is common path for hacker

So we can create custom bridge network by using which that container talk with host so the common path is broken

**Fifth pratical**

We can directly run container in detached mode

Docker run –d –name login nginx:latest

Login to container

Docker exec –it login /bin/bash/

Install on conatianer 🡪 apt update, apt-get install iputils-ping –y

Ping –V

Run another container

Docker run –d –name logout nginx:latest

Docker ps

Docker inspect containername

Docker exec –it logout /bin/bash/

Docker inspect containername

Ping logoutip

Docker network ls

Create bridge network 🡪 Docker bridge create secure-network

Docker network ls

Assigning secure-network to finance container

Docker run –d –name finance –network=secure-network nginx:latest

Docker ps

Docker inspect containername

Docker inspect logout

Try to ping finance from login container

Docker run –d –name host-demo –network=host nginx:latest

Docker ps

Docker inspect host-demo