Monolithic & Microservices Architecture

Monolithic

- It is all the component inside that are tightly coupled or it is a single block.

- Latency is low

- Have to write code in a single programming language

- Have to scale the entire API.

- Deploying is harder in Monolithic because if a program occurs we need to deploy the entire stack.

Microservices

- ~~It~~ Here all the component are separated Eg: Movie ticket booking

  user login
  booking ticket
  payment
  ] All are separate services functioning o single API.

- Here also latency is low and greater than monolithic.

- Have the flexibility to code in different languages. For each of the separated components.

- ~~Hee~~ Here we can scale or create an on the separate component which getting more load. Eg: In ticket booking app if ticket booking is getting more load we can scale that to 2 to 3 Folds So to solve the problem of heavy load

- Here we can deploy in the needed component ~~so there is no need to~~ and we can insert that part alone.

- Microservice has Full control of the API & it is getting more adaption.

Containers

Ans

- An Engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficent container that can be manipulated using standard operations and run consistently on virtually any hardware platform.

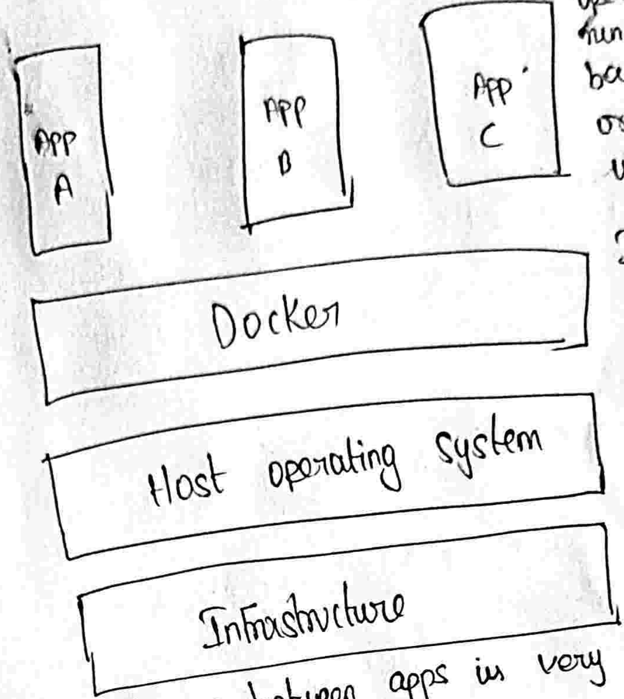
- It is like a box that contains our application and other stuffs that can be transferred within that box to another system & it will work efficiently in that system.

- It is like a software that contains all the package and code and other stuffs of our application it contains everything to run your application quickly from one environment to another.

- Need less resource of the hardware ie using less space in hardware

Containers Vs Virtual Machine's

Containerized Application

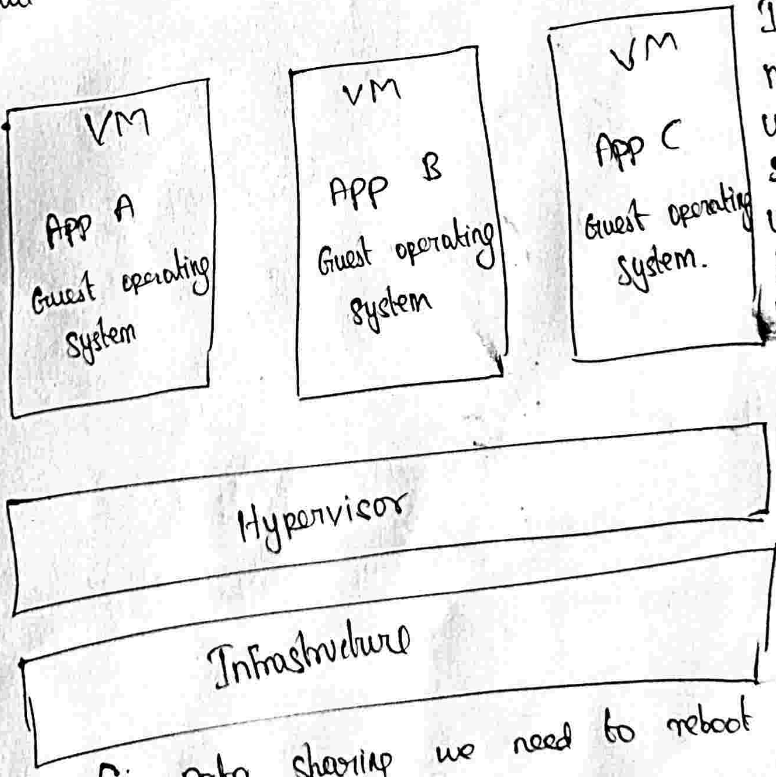


When we use containers instead of virtual machines the container act as a Host operating system & we can run any software that is basically a linux software or apple software or windows software

It takes very less space instead in virtual machine we need to do dual booting (ie) 8 windows & linux OS on same system this will take a lot of space in GB range.

Data sharing between apps is very efficient.

Virtual Machine



In one virtual machine of one OS sy is installed in ou system we can use ~~an~~ applicat that is based on another O for that we need to use two OS on same system This will lead to occupying more number of data's

for Data sharing we need to reboot to another operating system.

Practical Containers

Docker Files consists of application code, Library and dependencies.

demo -> cat Dockerfile

```
                              ↗ File in docker.
FROM     python:alphine3.7 -> Base image from   we need the
                              docker file
COPY  ./app -> copy image  from  base to the new image
WORKDIR  /app
RUN pip install  Flask
EXPOSE 8080 -> The port that has to be exposed during run time
ENTRYPOINT ["python"] cannot be overwritten.
CMD ["main.py"] // can be overwritten on run time
```

creating a docker file

```
                        ↗ account id
demo ~ docker build -t saiyam911/dockerfile-demo:V1
                                                 ↳ tag
```

we need to mention to otherwise it take the updated tag.

View image on System

```
docker image LS
```

Run a container in the image

```
docker run -it -P 8080:8080 saiyam911/dockerfile-
           ↓                       demo:V1.
      interactive
         mode
```

In this it will work once a time and if we refresh it will not load This is interactive mode.

detached mode

```
docker run -d -p 8080:8080 saiyam911/dockerfile-demo:v1
```

-d ↓ detached

It will work many times.

A python image or application is converted into a image in docker and it runs in container . . .

Docker Volumes

demo ~ docker run -it -v /home : /tmp/busybox

-it ↓ interactive

-v ↓ volume

demo ~ docker run it -v demo : /usr /share busybox

↳ name of the docker volume

/# cd /usr/ share /

/usr /share #LS → listing

/usr/ share # touch demo → creating demo

/usr/ share #LS → now listing

demo → demo is created so it is showing

/usr/ share /exit

View the volume

docker volume LS

	local
Driver	demo
local demo	

named volume.

To see where demo is created

cd / var/ Lib/ docker /volumes/

← current directory

→ subdirectory

Creating docker Volume without naming.

docker run -it -v /home busybox

/ # cd/ home -> going to the directory

/home # ls -> listing

/ home # touch lest 2 -> adding lest 2 volume

/home # ls -> listing

test 2 -> lest 2 is added so it is showing here.

/home # exist -> existing.

demo ~ docker volume ls

Driver   Volume Name

Local   36d550bd06e54d1 . . . ↔ random Volume is created.

Local   Demo -> old one

demo ~ cd /var/ lib/ docker /volumes/

demo - ls

36d550bd06e5 . . . . demo.

Usually named Volumes are mostly used