**Agenda:**

1. **Intro of Docker**

* What is Docker?How it works
* Containers
* Why we use it?

1. **Benefit of Containerization**
2. **Docker Arch.**

* Docker Engine, Components of docker
* Real world eg’s.
* VM vs Docker

1. **Creating Docker hub acc.**
2. **Starting to work with images**
3. **Docker commands**
4. **Dockerfile**

**Introduction to Docker**

**What is Docker?How it works**

**-**Open-sourced

-Automates the deployment of the application inside a light weight, portable container

-Simplifies the development, shipping, and deployment by giving the great environment across different systems.

Eg. Ubuntu size reduced to 78 mb from 4gb.

**Containers**

It is light weight, executable package, that includes everything needs to run a product(software) inc. code, runtime, tools, plugins, dependencies etc etc.

**Features:** LightWeight, Portable : across diff environment, Fast-Deployment, Scalability

| **How it reduces the size? Activity(5 mins)**  docker images contains only minimal OS dependencies  its an image.. from dockerhub  Docker images contain only the essential binaries and libraries required to run the environment..  Minimal base image,layered architecture  Through shared OS kernel,layer image.. it helps to reduce the size  in docker we use multi-stage builds  docker os image of reduce size  multiple container using from a single docker image |
| --- |

**Why we use it?**

| **Aspects** | **VM** | **Container** |
| --- | --- | --- |
| Architecture | Runs multiple OS by hypervisor | Shares the host OS kernel, runs **isolated** process |
| Size | Large size (GB) | Light weight, ( MBs), shared kernel, os lib |
| Speed | Slow - due to os boot time | Fast |
| Resources | Req, RAM and CPU | Less resources req., more efficient |

**Benefit of Containerization**

**1.Consistency :** Works the same on different env. Testing, prod, and env.

**2.Isolation**

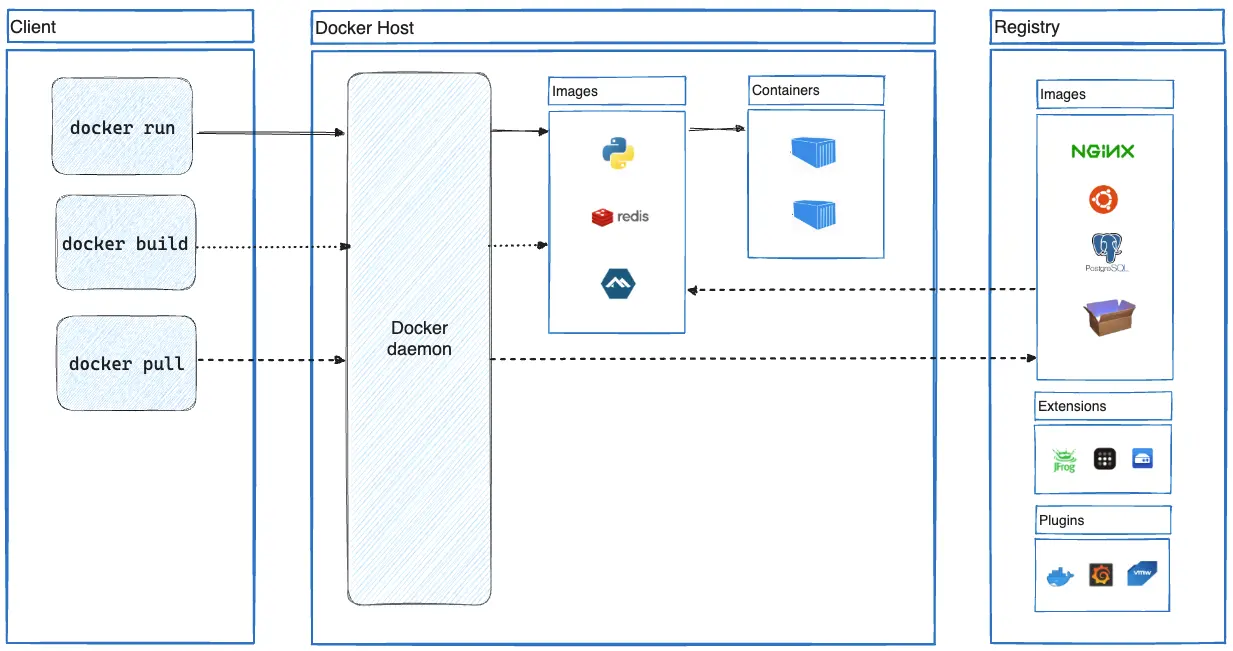
**3.Efficiency**

**4.Scaling on req.**

**5.Easier Deployment**

**6.Speed**

**Docker Architecture**

****

**Docker Engine**

Core component of Docker, manages the containers and containerization and networks

Consists of:

1. Docker Daemon: Run in background, manage images, containers, etc
2. Docker Client : CLI tools used to interact with the daemon
3. REST API : To communicate with the external tools nad channels with docker daemon.

**Docker Components**

1. Docker Client : Sends the commands to Docker daemon, tells what to do!

Eg. docker run nginx - starts nginx container

1. Docker Daemon : Handles all tasks related to container.
2. Docker image : Pre-built templates used to create containers.

Eg. Docker pull ubuntu

Eg. Downloading games from App store and downloading images from docker hub

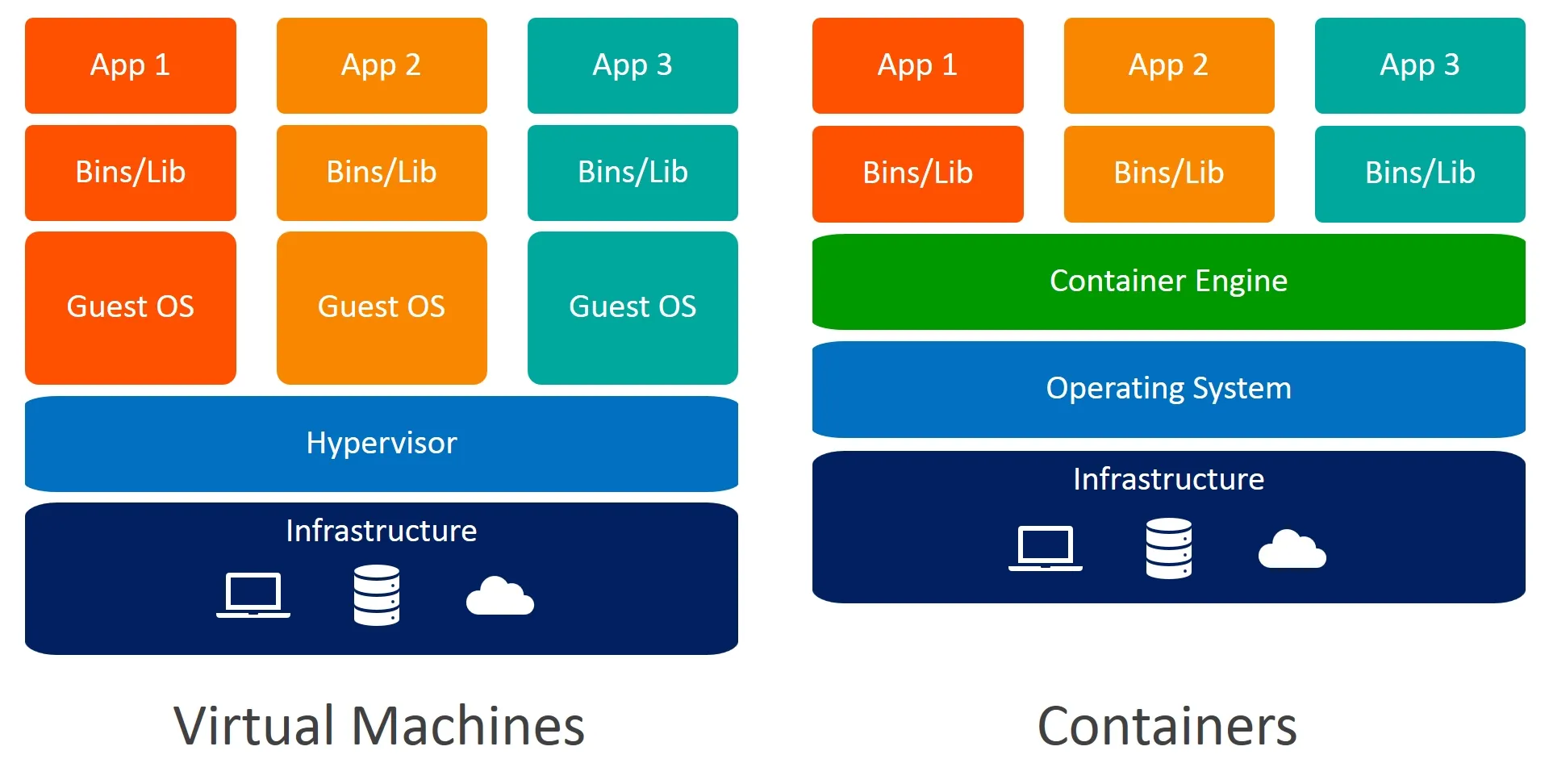
4. Docker Containers: Running the docker image

Eg. Like cargo containers, all are same but have different stuff like iphones, wheat, cars, gold but still are same from outside, similarly docker container are also having different images running but are same from outside!



5. Docker Registry : Stores Docker images (e.g. Docker Hub, or store privately)

**VM vs Container Architecture**



**Docker commands**

1. Sudo apt install docker.io -y
2. Which docker
3. Docker –version
4. sudo docker pull ubuntu
5. sudo docker images
6. sudo docker rmi 97662d24417b
7. sudo docker run -itd ubuntu
8. sudo docker ps : only running container
9. sudo docker ps -a : all containers
10. sudo docker stop 0679e287da52
11. sudo docker restart 0
12. sudo docker rm 36910cde614c
13. sudo docker rm -rf 36910cde614c for running container
14. sudo docker exec -it 87de6bf46ab5 bash go inside container
15. sudo docker run -itd -p 81:80 ubuntu

20-2

1 sudo apt update

2 sudo apt install docker.io -y

3 docker --version

4 sudo docker pull ubuntu

5 sudo docker images

6 sudo docker run -itd ubuntu

7 sudo docker ps

8 sudo docker run -itd -p 81:80 ubuntu

9 sudo docker ps

10 sudo docker run -itd -p 81:80 ubuntu

11 sudo docker ps

12 sudo docker run -itd -p 87:80 ubuntu

13 sudo docker ps

14 sudo docker exec -it 0b40 bash

15 sudo docker ps

16 sudo docker exec -itd bfc bash

17 sudo docker exec -it bfc bash

18 curl 18.217.39.68:87

19 curl 18.217.39.68:81

20 sudo docker ps

21 sudo docker exec -it df028cb64131 bash

22 curl 18.217.39.68

23 curl 18.217.39.68:80

24 sudo docker run -itd -p 85:80 ubuntu

25 sudo docker exec -it 399 bash

26 curl 18.217.39.68:85

27 sudo docker exec -it 399 bash

28 sudo docker run -itd -p 83:80 ubuntu

29 sudo docker exec -it ed9f bash

30 sudo docker ps

21-02

A-1  
Container commands  
  
 1 apt update

2 apt install apache2 -y

3 service apache2 start

4 service apache2 status

5 history

Vm commands

1 sudo apt update

2 sudo apt install docker.io -y

3 sudo docker pull ubuntu

4 sudo docker run -itd -p 80:80 ubuntu

5 sudo docker ps

6 sudo docker exec -it a4cadef92fb4 bash

7 history

A-2

Container  
  
 7 service apache2 status

8 service apache2 start

9 service apache2 status

10 history

VM  
  
 8 sudo docker ps

9 sudo docker commit a4cadef92fb4 samayimg

10 sudo docker images

11 sudo docker exec -itd -p 81:80 samayimg

12 sudo docker run -itd -p 81:80 samayimg

13 sudo docker ps

14 sudo docker exec -it bb bash

15 history

A-3

M-1

16 sudo docker ps

17 sudo docker images

18 sudo docker login

19 sudo docker images

20 sudo docker tag samayimg nikkhildock/samayimg

21 sudo docker images

22 sudo docker push nikkhildock/samayimg

23 history

M-2

1 sudo apt update

2 sudo apt install docker.io -y

3 sudo docker pull nikkhildock/samayimg

4 sudo docker images

5 sudo docker run -itd -p 80:80 nikkhildock/samayimg

6 sudo docker exec -it 61cd bash

7 history

**AGENDA**

What is dockerfile?

Why do we need it?

How its container automation works?

Commands/Directives

**What is dockerfile?**

A docker file is a script,

* Contains list of task or the instruction,
* That is required for building docker image
* It automates the setup of an environment.

**Why do we need it?**

1. Automation - need not to work manually i.e. on manual setup
2. Portability - Runs the same way on any machine
3. Consistency - Runs everywhere,”eliminated” it works on my machine”
4. Scalability - Easily deployable

**How its container automation works?**

Instead of manually running commands → dockerfile automated the process → on execution → command run → in sequence → creation “repeatable environment”

**Commands/Directives**

1. **FROM**

* Every dockerfile start With FROM, used to mention the base image.
* i.e.

1)FROM ubuntu

2)FROM ubuntu:latest

3)FROM ubuntu:22.04

1. **RUN**

* It is executed during the build
* Used to install software and dependencies inside **container**
* i.e

1. RUN apt-get update
2. RUN apt-get install apache2 -y

**3. CMD and ENTRYPOINT**

* Both define the command that runs inside the container.
* i.e.
* CMD [“echo”, “Hi this is samay”]
* ENTRYPOINT [echo”, “Hi this is samay, THIS WILL ALWAYS RUN”]

### **Which to use?**

* CMD - command can be changed during runtime.
* ENTRYPOINT - Wants specific commands to execute

**4. COPY and ADD**

* COPY: It copies the file,
  + cannot use url
* ADD: It copies files and extract the archive
  + When working with compressed files(e.g. .tar, .zip) you might need to extract them.
  + Can use URL

**5. ENV and ARG (Defining the variable)**

### **What is variable? Nd why we use it?**

* A variable in docker file is the value, that can is used to make it more **configurable**, It helps in setting up **environment settings, version, credential, build.**
* a key-value pair that allows you to pass configuration data to a containerized application at runtime, enabling flexible customization without modifying the underlying Docker image itself
* configure containers dynamically without modifying the image.

### **ARG(Build-time variable)**

* Defines the variable that is only available while building the image
* Once container is running its not accessible.
* Mainly used when need to pass custom value/
* I.e.
* FROM ubuntu:latest
* ARG APP\_v=1.0.0

### **ENV(Runtime Variable)**

* ENV defines the environment variable that persist inside the **running container.**
* Available to all processes inside the container
* Used for configuration like API keys, database url, or credential.

**6. Other Commands**

WORKDIR /app - sets the working directory

EXPOSE 80 - port

USER nginx - run the container as a specific user

26-02

FROM ubuntu

RUN apt-get update

RUN apt-get install apache2 -y

EXPOSE 80

ENTRYPOINT ["apachectl", "-D", "FOREGROUND"]

sudo docker build -t ubuntu:latest .

Assignments -5

1 sudo apt update

2 sudo apt install docker.io -y

3 docker --v

4 docker --version

5 sudo docker pull ubuntu

6 sudo nano Dockerfile

7 sudo nano index.html

8 sudo nano Dockerfile

9 sudo docker build -t custompg .

10 sudo docker images

11 sudo docker run -itd -p 80:80 custompg

12 history

FROM ubuntu:latest

RUN apt-get update

RUN apt-get install -y apache2

COPY index.html /var/www/html/

EXPOSE 80

ENTRYPOINT ["apachectl", "-D", "FOREGROUND"]

**Docker Compose**

1. What is Docker compose?

It is a tool that helps **manage multi-container applications** using single YAML file(**docker-compose.yml**):

What this file does?

* Defines services, network in structured way
* Manages dependencies between container,
* Scale services easily.

1. Why To use Docker Compose?

| Features | Advantages |
| --- | --- |
| Multiple container setup | Manage multiple containers as a single unit |
| Simpler commands | No need for long docker run commands |
| Environment variable | Easily config using .env files |
| Scaling | Scaling up/down containers is possible |
|  |  |

Basic usage compared with the Docker CLI

Basic usage:

Docker network create mynet

Docker run -d app –network mynet -p 83:80 myapp

**With Compose:**

version: “3.8”

services:

db:

Image: asdcdvdv

app:

image: myapp

Ports:

* “83:80”

To run:

docker-compose up -d

**Discussing Compose.yaml file:**

Ex: Nginx container

version: “3.8”

services:

web:

image: nginx

ports:

* “8080:80”

To run:

docker-compose up -d

**1: version**

* Specifies the docker compose file format version
* Different version supports different features
* Latest version is : 3.8

**2: services:**

* Defines all containers that will be deployed
* Each service represents a container
* Includes Configuration: image, port, dependencies. Etc.

**3: Volume & network:**

* Volume : store data :eg database storage
* Network : how containers communicate.

Ex: Nginx container

version: “3.8”

services:

web:

image: nginx

ports:

* “8080:80”

volumes:

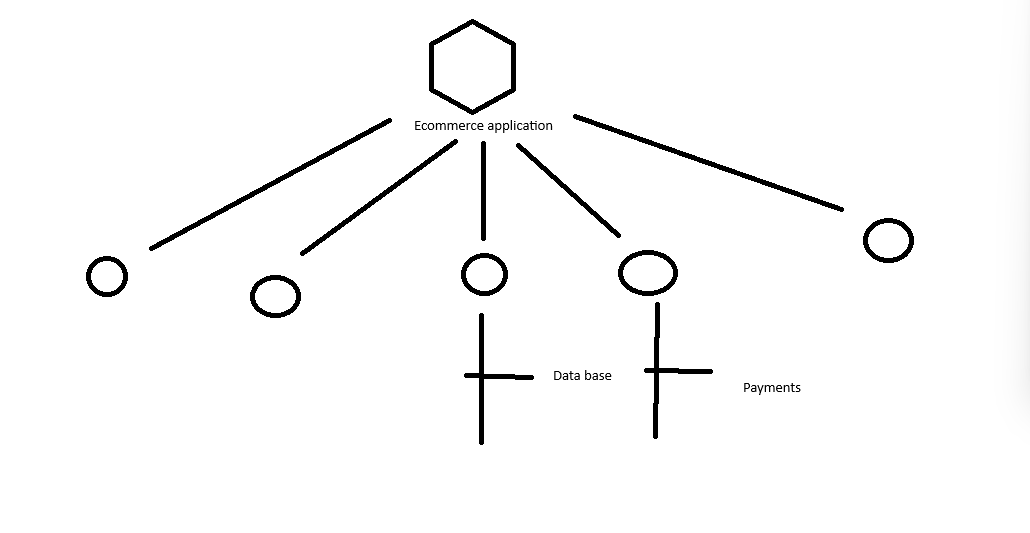
* .html:/fsbvsf/dvsdv/dsbsfdb/sbvsd

To run:

* docker-compose up -d

To stop and remove containers:

* docker-compose down



**Working with volumes and Networks:**

- Docker Volume → allows persistent storage – even if the container stops

Eg.

version: “3.8”

volumes:

pgdata:

services:

db:

image: postgres

volumes:

* pgdata:/var/lib/postgressql/data

**Commands**

docker volumes ls

docker volume inspect pgdata

**Networking:**

By default docker network creates services where services can communicate using service names.

**Example : Custom Network**

version: “3.8”

networks:

mynetwork:

services:

db:

image: postgress

networks:

* mynetwork

web:

image: myapp

networks:

* Mynetwork

Mynetwork: ensures isolation

Db and web can communicate via service names

**How to check?**

Docker network ls

Docker network inspect mynetwork

**Hands-On :**

You are working as a DevOps Engineer, in a company that is building a web application, using NGINX as a web server and MYSQL as a data base. Your task is to containerize this application and manage multiple service using Docker Compose.

Goals:

* Use Docker Compose to define and run the multi container application
* Deploy the nginx based appltn, that connects you to a mySQL data base
* Ensure that the service communicates within the docker network.
* Expose to the port 8080.

Working with database :

Root password

data base

User

Password

version: '3.8'

services:

web:

image: nginx

container\_name: nginx-web

ports:

- "8080:80"

volumes:

- ./nginx/html:/usr/share/nginx/html

depends\_on:

- db

db:

image: mysql:5.7

container\_name: mysql-db

environment:

MYSQL\_ROOT\_PASSWORD: rootpass

MYSQL\_DATABASE: testdb

MYSQL\_USER: user

MYSQL\_PASSWORD: password

volumes:

- db\_data:/var/lib/mysql

volumes:

db\_data:

Commands   
 1 sudo apt update

2 sudo apt install docker.io -y

3 sudo apt install docker-compose

4 docker --version\

5 docker --version

6 docker compose --version

7 history

8 mkdir dockercompose

9 ls

10 cd dockercompose/

11 ls

12 touch docker-compose.yml

13 ls

14 nano docker-compose.yml

15 cat docker-compose.yml

16 cd

17 mkdir -p nginx/html

18 ls

19 cd nginx/

20 ;d

21 ls

22 cd html/

23 ls

24 nano index.html

25 cd

26 docker compose up -d

27 docker-compose up -d

28 ls

29 cp nginx/ dockercompose/

30 cp -r nginx/ dockercompose/

31 ls

32 cd dockercompose/

33 ls

34 docker-compose up -d

35 sudo nano docker-compose.yml

36 docker-compose up -d

37 sudo docker-compose up -d

38 docker ps

39 sudo docker ps

40 sudo docker exec -it mysql-db mysql -u root -p

41 sudo docker-compose down

42 sudo docker volumes

43 sudo docker volume

44 sudo docker-compose down -v

45 history