Chapter - 4 Docker

4.1

Intro :-

→ Docker is an open-source platform that devs use to package softwares into standardized units called containers.

→ Docker is a platform for developing, shipping and running applications in containers.

→ Containers provide a lightweight and consistent environment ensuring that an application and its dependencies run consistently accross the different environments.

→ Docker facilitates the packaging of an application and at its dependencies into a single container that can run on any docker enabled system.

→ Docker container is a portable unit of software.

Use cases of Docker :-

1) Application Isolation :
Containers provide a way to isolate an application from the underlying system, ensuring that it runs consistently accross different environments. (Web + app) (4) CI/CD

2) Micro Services :-
Docker is often used in microservices architecture, where each microservice is deployed as a separate container, enabling easier scaling and maintenance.

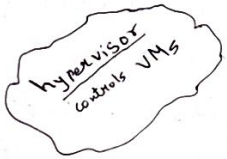
3) DevOps : Docker is popular in DevOps practices as it streamlines the development, testing and deployment processes.

Platforms for Docker :-

-> Linux, Windows and MacOS.

-> Docker Desktop provides an easy way to install and manage Docker on Windows, Linux and MacOS for development purposes.

Docker vs Virtualisation :



Virtualization is the process of making VMs.

- Virtualization :- In traditional virtualization, each VM includes a full OS which can lead to significant overhead in terms of resource consumption and startup time.
- Docker Containers : They share the host OS kernel and only include the application and its dependencies resulting in lower resource usage and faster startup times compared to VMs.

Problems Docker solves

1. Consistent environment
2. Dependency management
3. Isolation
4. Resource efficiency (share host OS)
5. Portability (containers can run on any system with Docker installed)
6. Scalability

Difference b/w VM and Docker

Feature	Docker	Virtual Machine
1. Compatibility	works best with Linux Distributions	All operating Systems.
2. Size	light in weight	larger (in GBs or TBs)
3. Virtualization	only the application layer	both the OS kernel and application layer.
4. Performance	easy to start containers (in milliseconds)	takes longer to start a VM instance
5. Security	less secure	relatively more secure
6. Replicability	easy to replicate. You can pull Docker images corresponding to the various applications.	difficult to replicate, especially with increasing number of VM instances.

4.2

Docker Architecture

16/03/024

*

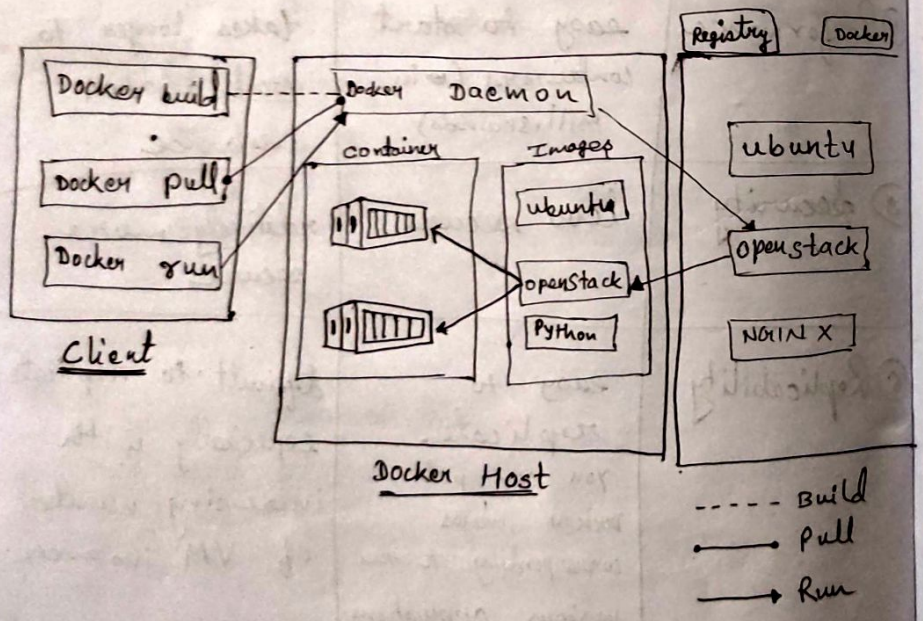
Q. Explain Docker architecture in Detail? (10 marks)
Also explain its components.

Ans

→ Docker uses a client-server architecture.

→ Docker's architecture is designed to provide an efficient and consistent way to package, distribute and run apps using containers.

→ It promotes isolation, portability and scalability while making it easier to manage complex application deployment.



The Docker components

$h^2 = p^2 + b^2$

① Docker Client :-

The Docker client is a command line tool or API interface that users interact with to manage docker container and images.
It sends commands to the Docker Daemon for execution.

② Docker Daemon :- (Docker Engine)

The Docker Daemon is the core background process responsible for building, running, and managing containers. It listens to Docker API request from the Docker client, handles Container Lifecycle, and manages the container's file system, networks and storage.

③ Container Images :-

Images are read only templates that define the applications and its dependencies. They include everything needed to run a Container, such as code, runtime, system tools, libraries and settings. Images are built from a set of instructions specified in a Docker File.

④ Docker Registry :-

* It is a repository that stores and distributes Docker Images. Docker Hub is a popular public registry, but you can also setup private registries.

You can push and pull images to and from registries to share and distribute container Images.

⑤ Docker Container :-

→ It is a runtime instance of an Image. It encapsulates the application and its dependencies, running in an isolated environment.

→ Containers share the host OS Kernel but have their own isolated file system, processes and ~~their own~~ network.

⑥ Docker Compose :-

→ It a tool for defining and running ~~multiple~~ multi-container Docker application. It uses a YAML file to define the services, networks and volumes needed for an application and then starts and manages those containers as a single application.

⑦ Docker Swarm :

It is a Docker's native clustering and orchestration solution.

It allows you to create and manage a cluster of Docker nodes, enabling features like load balancing, scaling, rolling updates and service discovery.

⑧ Docker CLI :

The docker command line interface is used to interact with Docker.

* (2) docker build :

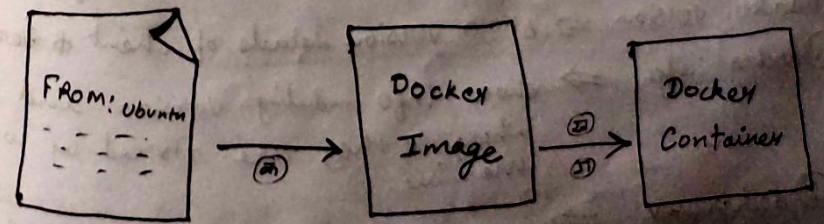
builds docker images from Dockerfile.

* (3) docker run :

runs containers from docker images.

* (4) docker push :

pushes the container image to public/private registries ~~to~~ share the ~~Docker Images~~ Docker Images.



Date 17/03/024

Q. Difference between Container and Image.

→ Container is a running environment for an image.

~~→ Applicati~~

Provisioning
The process of preparing and configuring a Docker container or image for use.

Docker Commands

① docker run ⇒ create and run a new container from an image.

② docker build ⇒ build an image from a dockerfile.

③ docker pull ⇒ download an image from a registry.

④ docker push ⇒ upload an image to the registry.

⑤ docker ps ⟹ lists containers

⑥ docker exec ⇒ execute a command in a running container.

⑦ docker images ⇒ list images in local machine.

⑧ docker rm ⇒ remove one or more ~~commands~~ containers

⑨ docker rmi ⟹ remove one or more images.

⑩ docker version ⇒ show version details of client & server.

⑪ docker info ⇒ show info including version and paths where they are stored in local machine.

⑫ docker -v ⟹ gives version of docker.

Practical (Piyush Garg)

1) `docker run -it ubuntu` (-it ⇒ interactive mode)

→ container "name" randomly देता है।

~~→ container terminal से exit~~

→ अगर terminal, container से बाहर run करने लगेगा।

→ ctrl+d ⇒ exit Container.

This command checks whether the "Ubuntu" images is present locally or not.

If present :

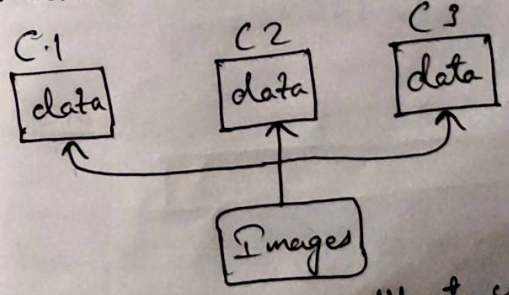
Create a container by using that image and opens the terminal inside the container.

else :

Downloads the images from "hub.docker.com" then it creates container, opens terminal inside it.

Docker = images + container

→ An image can run in multiple Containers



This command will start an Ubuntu container and provide an interactive bash shell, allowing to execute commands inside the container as if working on a Linux terminal

Some more docker commands :-

1. docker container ls ⇒ list running containers
2. docker container ls -a ⇒ list all present containers
3. docker start "containername/containerId"
4. docker stop "containername/containerId"
5. docker login
6. docker network inspect bridge

-it → -i for interactive and -t for -tty.
-it allows you to interact with a docker container as if you were working directly on the command line inside the container

* **docker run ubuntu** :-
It will create new container of Ubuntu image, runs the image inside the container and finally stop the container and returns back to local terminal

Syllabus : Container Routing

PORT MAPPING (Internal Working)

If you are running a service (eg - website) in a container, then you need to expose the port of container to the local machine's server port so that the website can be accessible by you using any web browser.

```
docker run -it -p 8080:8080 imageName websiteName
                C    LM
```

port mapping.

C = container port
LM = local machine port

ENVIRONMENT VARIABLES

```
docker run -it -p 1025:1025 -e key=value -e key=value
    websiteName
```

18/03/024

Containerize ① Dockerization of Node.js Application

1. Create a folder "docker-node"
2. change directory "cd docker-node"
3. Open VS Code here using "code ."
4. Inside VS Code :
   - Open terminal
   - "npm init" ⇒ creates package.json
   - "npm i express" ⇒ Installs & Express and node module get created.
   - create "main.js" and write required code.
5. Creation of "Dockerfile"
   ⇒ Dockerfile is a extensionless file which is used to write configurations about Image.
6. Conversion into Image :
   - syntax ⇒ docker build -t(tag ई) ImageName ~~locate~~ Dockerfile_ka_path
   - 'docker build -t mera-nodejs-image'
7. Run Image inside Container
   - `docker run -it imageName`
     ⇒ it'll create the container but ~~data will not~~ localhost:8000 will not work
   - now, port mapping is needed to be done between container port and local machine port, so that it can be accessible through browser.
   - `docker run -it -p 8000:8000 imageName`
8. `docker exec -it container_ka_id bash`
   ⇒ opens a new bash terminal inside the container whose id is specified in the above command.
9. 'ls' ⇒ list all available files of the container.
10. 'cat main.js'
    ⇒ cat ~~com~~ show the content of main.js in the terminal.

Publishing Custom image to DockerHub.

→ Go to DockerHub and Login.
→ Go to Repository tab.
→ create a repository
→ on the RHS, a command is visible, copy the "user/imageName" from that command and ~~com~~ locally create image of that ~~nam~~ copied name.
→ Open new terminal and write "docker push user/imageName"
⇒ it will give error, if you are not logged In to DockerHub

To login using ② Terminal :

→ docker login

⇒ username : *give user Name

⇒ password : give it

① Docker Compose

→ when we do real world development, we usually have multiple containers like one for ~~post~~ mongodb, one for redis, etc.

→ Each container has it's own configuration, port mapping and others. You need to write these commands somewhere and run them line by line which is NOT A GOOD WAY.

→ If there are multiple containers, you need to ask which containers are to run. (Problem है)

⇒ So the solution is :- **Docker Compose**

→ Docker compose can be used to create, ~~run~~ setup and destroy multiple containers.

Eg :- let's do a practical.

① create "docker-compose.yml" file ~~in root of~~

② write the required code in it.

③ "docker compose up" ⇒ जितने भी configurations "docker-compose.yml" में दिए हैं, सब को up & running कर देगा।

⇒ downloads & runs given services (image, here) on the local machine.

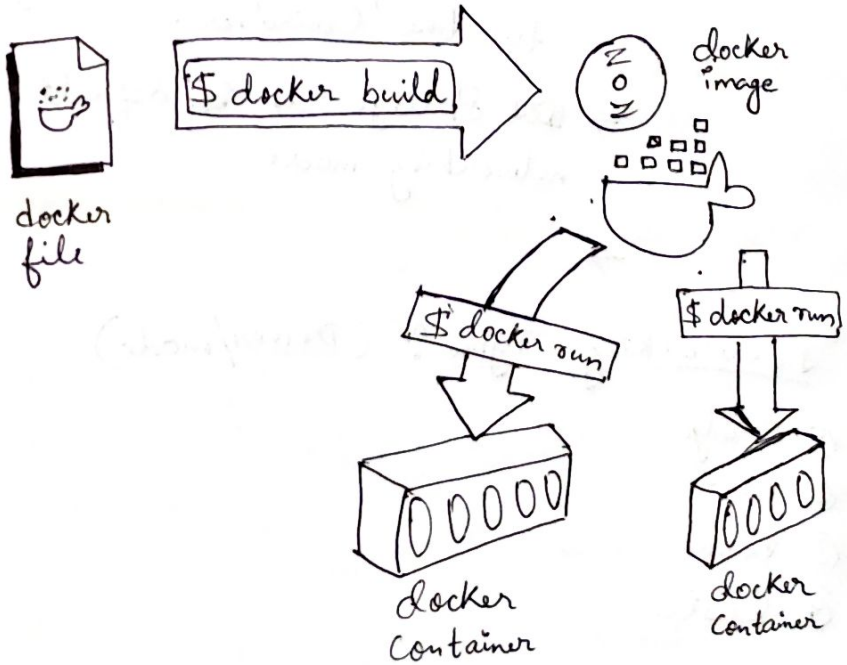
④ ctrl+C ⇒ exits all running containers.

⑤ "docker compose down"

⇒ सारे चल रहे containers को delete कर देता है।

⑥ "docker compose up -d"

⇒ runs the services in detached mode i.e in the background (without cmd).



Working of ✿ Docker.

19/03/024:

Docker Networking

`docker run -it --name my-container ImageName`

↓ we can manually give container name, if not random name will be given.

⇒ **Bridge** :- → ~~A~~ type of networking modes which gives the internet access facility.

→ Establishes bridge between host machine and container which provides internet facility to the Container.

→ ~~Do~~ Bridge is ~~by~~ default networking mode

→

Networking Types :- (Driver/mode)

1. Bridge
2. Host
3. None
4. Custom

→ This newly created container's name will be "my-container"

→ If you do "`ping google.com`". It starts communicating with google.com

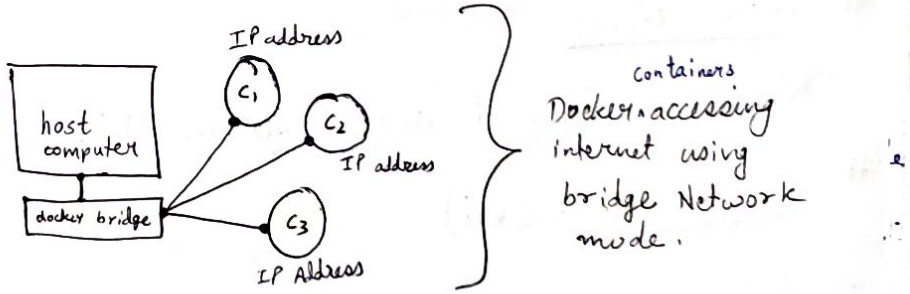
→ This shows that the container can ~~co~~ access internet. Docker establishes ~~it~~ a connection called **Bridge** (by default) b/w our local machine and container. This Bridge helps the container to access internet.

→ There are 3 types of Network connections in Docker

⇒ Bridge (default) ⇒ Host ⇒ None

→ You can also create your own Network i.e custom network.

→ `docker network inspect bridge`



→ `docker network ls` => lists all available ~~ntwrs~~ networks.

Host Mode

```
docker run -it --network=host busybox
```

→ In host mode, the container doesn't get a bridge, it gets directly connected to Host machine network. (खतरा ज्यादा security threat)

→ So, here port mapping not needed.

```
docker run -it -p 3000:8000 nodejs   X
docker run -it nodejs                ✓
```

--network=host

→ जिस भी port में Nodejs का ~~app~~ server चल रहा होगा automatically वो local host में available होगा because host machine और docker container same network पे है / so not needed to give port mapping

None Mode

Removes internet access from this container.

Custom Mode (vvi)

→ docker network create [-d] bridge networkName
(-d → driver)

⇒ creating a new network of bridge type.

⇒ `docker network create -d bridge youtube`

→ using the newly created network: youtube

⇒ `docker run -it --network=youtube --name tony-stark ubuntu`

↳ A container named tony-stark is created using ubuntu image and is using youtube network.

ubuntu में ping पहले से install नहीं रहता है ;

→ open new cmd and create a new container using busybox image.

⇒ `docker run -it --network=youtube --name dr-strange busybox`

→ Since both containers (tony-stark & dr-strange) are connected to same network, so they can communicate with each other. How ?

Ans: ⇒ go inside tony-stark container. ~~and ping~~ other install ping.

⇒ and ping dr-strange.

→ Since both containers are connected with same network, ~~therefore~~ no need of IP address to ping. Just use containerName/containerId instead of IP.

⇒ IP address may change over time but ~~containerName~~ hostname (containerName) doesn't.

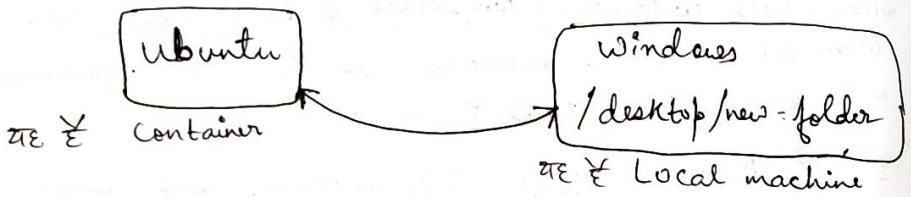
⇒ `docker network rm youtube`

This command will delete custom network "youtube".

Volume

Every container has its own memory but when we delete container, the memory also gets deleted.

Mounting a directory (local machine) to a container (Volume Mapping)



Let Suppose we have two containers, one is ubuntu and another is busybox.
As we know that containers are isolated so they cannot communicate with each other & on deleting, their data also gets destroyed. So sol^{n} is -
To make communication possible between them, we will use mounting concept, where a particular directory from local machine (windows) is mounted to both containers.
This basically means both containers can access that directory and data gets permanently stored on that directory.

How to do mounting?

volume

```
docker run -it -v localMachineDirectory:ContainerDirectory ImageName
```

LocalMachineDirectory पे GitBash open करके pwd करो, use that result in above command.

Advantage of Volume :-

⇒ Volume doesn't increase the size of container

⇒ Volume means directory of local machine mounted to a container.

Efficient Caching of Layers

After building container from a custom image, if we modify any file from image directory then docker file starts executing from that particular layer which we modified in the Dockerfile.
So, while making layers in the Dockerfile we need to carefully place the commands.

read more about caching