DOCKER :-

DOCKER :

DOCKER is a Containerization tool Applications will be running inside the container

Docker is a open source platform designed to simplify the process of developing, shipping and running applications using Containerization.

Containers allow developers to package an application along with all dependencies {libraries, Configurations etc.} into a single, light weight, and portable unit that can run Consistently across different environments.

Why use docker :-

portability :- Runs Consistently on any system (Local, Testing)

Isolation :- Each Container is isolated from others and the host system.

Efficiency :- Uses Fewer resources than traditional VMs,

Fast deployment :- Containers start almost instantly.

Simplified CI/CD :- Docker Integrates well with Devops pipelines for automated testing/deploys.

Application types:-

* Monolithic
* Micro Service

Monolithic:-

This application is built as a single, tightly-coupled unit. All components - like UI, business logic, and database access - are packaged and deployed together

Example:-

A Traditional E-Commerce Site

* product Catalog
* order processing
* User authentication
* payment System.

Micro Service:-

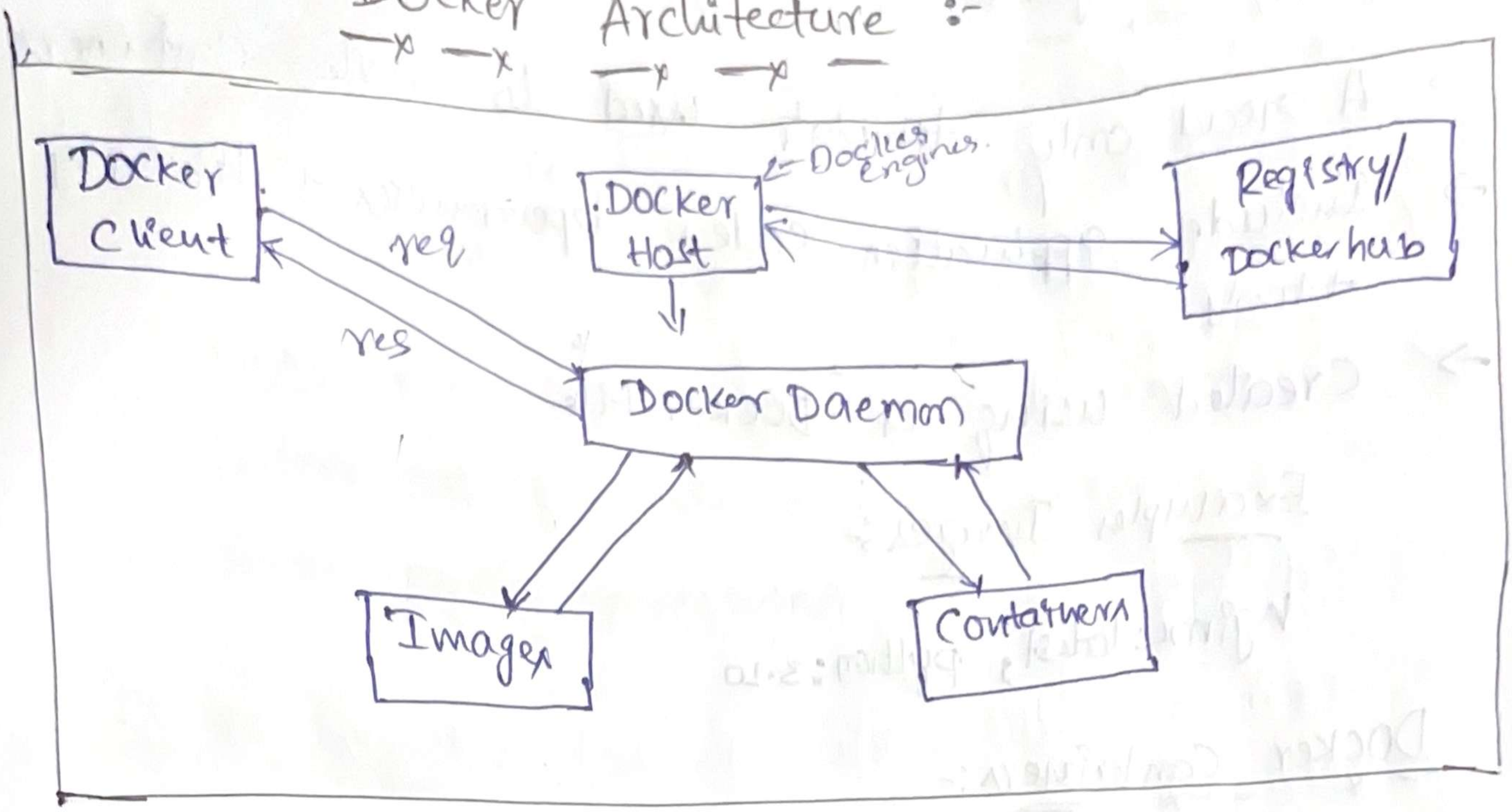
This application built as a collection of small, Independent services.

Each service focused on specific business Function and can be developed, deployed, and scaled Independently

Example:- E-Commerce platform.

* A product service for catalog management
* An order Service for order handling
* A User Service for authentication
* A payment Service / Each can Independently (HTTP/REST

Docker Architecture :-



Docker Client :-

→ The user Interacts with Docker using Commands like docker build, docker run, etc

→ sends these Commands to the docker daemon via REST API.

→ The client can Communicate with local (or) remote Docker daemons.

Ex :-

Docker run nginx.

Docker daemon :-

→ Core engine of Docker running in the background

→ listen docker requests from the client

→ Responsible for building, running and managing Containers and Images.

Docker Images :-

→ A read only template used to create containers
→ Include application code + dependencies + libraries + tools
→ Created using a "Docker file"

Examples Images :-

nginx : latest, python : 3.10

Docker Containers :-

→ A run time instance of a Docker image
→ Containers are light weight, isolated, and run the application
→ Created Docker images using docker run.

Example :-

Running a Node.Js app Inside a container

Docker File :-

→ A text file with step-by-step instructions to build a Docker image.
→ Include Commands like FROM, COPY, RUN, CMD.

Ex :-

```
FROM python:3.9
COPY ./app
RUN pip install -r requirements.txt
CMD ["python", "app.py"].
```

Docker registry :-

→ A Central place to store the images and share
→ Docker hub is the default public registry.
→ you can also set up private registery.

Example :-

Do

Docker pull ubuntu

docker push myapp:latest.

Docker Engine :-

→ It combines the Docker client + daemon + Container runtime
→ It is the Full-stack responsible for building, running and managing Containers.

Components Interaction Flow :-

→ User runs a Command via Docker CLI
→ Docker CLI Sends the request to Docker ~~registry~~ Daemon if not available locally.
→ Docker daemon pulls image from Docker registry
→ Daemon creates and starts Container from Image.
→ Container runs as an Isolated process on the host OS.

Docker Compose :-

Docker Compose is a tool defining and running Multi-Container Docker applications using a simple YAML file (docker-compose-yml)

→ When you use multiple services like web + database cache, compose makes it easy to manage them together

→ Defines services, Networks, and Volumes in a Single YAML file.

→ Easily start all services with docker-compose up.

docker Compose yaml.

Example :-

```
Version : '3'
services :
  web:
    images : nginx
    ports : 8080

  db:
     images : mysql
environment :
  · MYSQL_ROOT_PASSWORD : Secret.
```

Docker Swarm :-

Docker swarm is Docker's built-in-tool for clustering and orchestrating Containers across a group of machines (nodes)

→ High availability and Load balancing
→ Service Scaling
→ Secure by default with TLS encryption.
→ Can turn Multiple docker hosts into a Single Virtual docker host

Example:-

```
docker swarm init
docker service create --replicas=3 --name web nginx
```

Containerization :-

Containerization is a light weight Method of running applications in isolated user spaces Containers on the same operating system Kernal

Key characteristics :-

→ Lightweight no guest OS required
→ portable Can run anywhere Docker is installed
→ Consistent environment across dev, test, prod.

Virtualization :-

Virtualization is the process of creating virtual machines that run on top of a hypervisor like (VMware, VirtualBox, or Hyper-v) Each VM has a full OS

Key Characteristics :-

→ Each VM includes its own kernal and os
→ Heavy Weight and slower to start Compared to Containers.
→ Better Isolation and Security between Environments.

/var/lib/docker :- Default path of docker.

DOCKER Commands

Container Commands :-

docker run -d -p 8080:8080 nginx : Run a container in detached Mode

docker exec -it Container name /bin/bash : to enter the container Inside.

docker start Cnt name : start a stopped Container

docker stop Cont name : stop a running container.

docker ps : List running containers.

docker ps -a : List all Containers {stop/start}

docker rm Cnt-name : Remove a container

docker logs : View Logs of Container.

Image Commands :-

docker pull nginx : pull image from docker hub.

docker build -t myapp:v1 . : Build an Image from a docker file

docker images : List images.

docker rmi nginx : Remove an image.

docker push myrepo/myapp:v1 :- push an image to a registry

Volume & Network :-

docker volume create myvol : create a volume

docker volume ls : list volumes

docker network create mynet : create a Custom network

docker network ls : List networks

System info & Cleanup :-

docker stats : Show resource usage

docker info : show docker system info

docker system prune : clean up unused data

docker volume prune : Remove unused volumes

Miscellaneous :-

docker Login : : Login to docker hub

docker Logout : : Logout from docker hub.

docker search nginx : : search docker hub images.

docke inspect my container : : show low-level info.

Docker Volumes :-

Named volume :-

Create explicity with docker volume create or automatically when specified in a container

→ Managed by docker and stored in dedicated directory.

Anonymous volumes :-

→ Created automatically when a container specifies a volume but doesn't name it

→ Get a random hash as their name

persistant data

Bind volume :-

It allows you to mount a specific file or directory from the host machine into a Docker container. This enables the container to read from and write to files on the host system directly.

Docker Networks :-

Bridge Network :-

→ Default network created when Docker is installed {named "bridge"}

→ Create a private internal network where containers Can Communicate

Host Network :-

→ Remove network isolation between container and host

→ Container uses host's Network directly.

* No need network installation
* When you need maximum performance.

None Network :-

→ Container has no access

→ only has a loopback interface {no external activity}

* Completely isolated containers
* Security - Sensitive applications

Some real time Commands:

→ docker run --name Jenkins -d -p 8080:8080 jenkins/jenkins:jdk1

Create Container name with Jenkins Image

docker Inspect Jenkins/Jenkins: To check default network.

How can I run a Container in host network?

docker run --name S2 -d -p 8080:8080 --network host Jenkins/Jenkins:JDK17

{ Remove -p and port number }

Two Containers Connectivity :-

→ ping Container IP

→ first docker exec

How to delete all Containers start/stop

docker rm -f $(docker ps -aq)

→ Create two Containers with 2 Images.

S1 → Jenkins

S2 → sonarqube

→ DOCKER Inspect S2

→ Copy Ip address

→ enter another Container S1→

→ docker exec -it c1 /bin/bash

Enter the normal root User.

↓

docker exec --user root -it Jenkins /bin/bash

→ apt update -y

→ apt install

→ apt install iputils-ping. -y.

→ ping IP address of S2.

→ exit

Dis-Connect from Bridge network:

bridge → bridge

bridge → none

none → bridge

* . You cannot Connect or disconnect to a host network

→ docker network --network help.

* docker network disconnect bridge Jenkins

* docker network Connect none.

Create bridge network:

docker network create --driver=bridge siva-bridge

docker run --name T3 -d -p 8080:8080 --network siva-bridge image name

Docker system info: It is used for to see the Container data images data cpu and memory usage

→ docker network create --driver-bridge --subnet=192.168.0.0/16 192_network

Docker volume prune:- It will be deleted unwanted volumes

docker system prune: clean up unused data

Create volumes:-

→ docker volume create myvol

→ docker inspect sivaa

Create path if delete Jenkins back up

* Create Container with J1 Jenkins
* Create JOb & Delete Container
* create directory in opt

→ docker run --name J3 -d -p 8080:8080 -v /opt/Jenkins:/var/Jenkins_home Jenkins/Jenkins:Jdk12

Derivaties in Docker file :-

```
# Use the official tomcat image
FROM tomcat:9-jdk17

# Copy the WAR file from the target directory into Tomcat's webapps folder
COPY target/*.war /usr/local/tomcat/webapps

# Apply permissions to the copied WAR file(s)
RUN chmod 755 /usr/local/tomcat/webapps/*.war

# Expose the Tomcat default port
Expose 8080

# Start Tomcat
CMD ["catalina.sh", "run"]
```

Security tools :-

1. Sonarqube as SAST ---- Static application security testing
2. Trivy :- It is mostly used for scan the docker images

Trivy :-

Installing process

* Go to repository microserviceapp
* clone trivy.sh

* cd microservice
* sh trivy.sh
* trivy --version.
* trivy image jenkins/jenkins:jdk17.

TOOL Name: Trivy

purpose: Security scanner for Container images and cloud-native artifacts

Scan for: vulnerabilities, mis configurations, Secrets.

Supported targets: Docker images, filesystems, repositories, Iac configs.

Integrations: Docker CLI, CI/CD tools, kubernetes.

Vulnerability :-

* Weakness in Software that can be Exploited
* can lead to attacks, breaches, data loss
* we can found on Docker images, OS packages, app dependency
* we can detect tools: Trivy, Syft, Grype
* how to fix: update components, apply patches, Follow best practices.

Difference between COPY and ADD in dockerfile :-

COPY :-
To copy files and directories from the build-context (host) into the Docker image

COPY Source destination

* only copies local files and directories
* Does not handle remote URLs.
* Does not extract archives (.tar, .tar.gz)

ADD :-
performs everything 'COPY' does, plus extra features.

* you need to extract archives or download from URL

Docker Compose :-

Docker Compose is a tool that allows you to define and manage multi-container Docker applications.

It uses a YAML file {docker-compose.yml} to Configure the Services Containers, Volumes and Networks your application needs, and manage them with a single Command.

* Defining Services in one file
* Managing Services together {up, down, restart}
* Ensuring Services start in the Correct Order.
* Supporting Environment Variables, build-Instructions, volumes and Networks.

Docker Compose Commands :-

docker-Compose up : Builds, recreates, and starts all Services

docker-Compose up -d : Run Services in the background (detached mode)

docker-Compose down : Stops and remove Containers Networks, volumes.

docker-Compose build : Builds/rebuilds Images

docker-Compose logs : Show logs from all Services

docker exec <Service Command : Runs a Command in a running Service Container

Docker Compose file structure :-

```yaml
Version: "3.8"

services:
  web:
    build:
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    depends_on:
      - db
  db:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
```

Docker - Java web calculator :-

→ clone microservices repo

→ cd micro

→ sh docker install.sh.

→ Clone Java web calculator. URL

→ Install OpenJDK.-17

→ apt update

→ sudo apt install maven -y.

→ Cd Java web

→ Java -- Version

→ mvn -- Version.

→ mvn package

→ sudo cat /etc/gshadow | grep 'docker'

→ sudo usermod -aG docker ubuntu

→ newgrp docker

→ mvn package

→ cd /target

→ Cat Dockerfile2

→ rm Dockerfile

→ mv Dockerfile2 Dockerfile

→ docker build -t myapp:v1 .

→ docker image

→ docker run --name mywebcalc -d -p 8081:8080
myapp:v1

→ copy Ip add
→ 8081:8080/webapp-0.1/
→

How to deploy the nodeJs application

→ git clone JavaWebCalculator.
→ go to aws instance
→ click RDS → for creating database
→ click. Aurora and RDS. → standard create
→ create database
→ click My sql
→ free tier
→ master password → Shiva@996325 /username - admin.
→ Connectivity
↓
Connect EC_2
→ create data base.

→ Add Security Mysql portnum:- 3306

database - Server.

→ sudo apt install mysql-client-core-8.0
→

Difference between CMD and Entrypoint :-

→ Both CMD and ENTRYPOINT are Dockerfile instructions that specify what command gets executed when running a container, but they have behave differently.

ENTRYPOINT :-

* Defines the executable that will run when the container starts.
* Specifies the main command of the container.
* Arguments passed to "docker run" are appended to the ENTRYPOINT command.
* If ENTRYPOINT is defined, CMD provides default arguments to ENTRYPOINT.
* Best for containers that should behave like an executable.

CMD :-

* provide default arguments for the ENTRYPOINT or default command to run.
* Can be easily overridden by passing arguments to "docker run"
* If no ENTRYPOINT is specified, CMD defines the command to execute.

Key differences :-

CMD :- Can be Completely overridden by Command line arguments to "docker run"

ENTRYPOINT :- It is harder to override (requires --entrypoint flag)

procedure :-

→ git clone Installation.
→ Install docker
→ Sudo usermod -aG docker ubuntu
→ newgrp docker
→ rm Dockerfile2
→ mv Dockefile2 Dockerfile