DOCKER

1. **Docker installation:**
    a. Sudo apt update
    b. Sudo apt install docker.io -y
    c. Sudo service docker start
    d. Sudo service docker status
    e. Sudo systemctl enable docker.service

2. **Switch to root user:**

    sudo su

3. **Pull the NGINX image:**

    docker pull nginx

4. **List all Docker images:**

    docker images # or docker image ls

5. **Search for a Docker image (e.g., MySQL):**

    docker search mysql

6. **Pull the MySQL image:**

    docker pull mysql

7. **Show the history of an image (by name or ID):**

    docker history <image-name or image-id>

8. **Run a Docker container from an image:**

    docker run <image-id or image-name>

9. **Run a container in detached mode:**

    docker run -d <image-id or image-name>

10. **List running Docker containers:**

    docker ps

11. **Access a running container's bash shell:**

    docker exec -it <container-id or container-name> /bin/bash

12. **Navigate to NGINX's HTML directory:**
    - cd /usr/share/nginx/html/
    - ls

13. **Update package list and install editors**
    a. apt update
    b. apt install nano
    c. apt install vim

1. **Remove a Docker container:**
    - docker rm <container-id> # or docker container rm <container-id>
2. **List running containers:**
    - docker container ls
3. **List all containers (including stopped):**
    - docker container ls -a
4. **Alternative way to list all containers (including stopped):**
    - docker ps -a
5. **Inspect container details:**
    - docker inspect <container-id or container-name>
6. **Test container access using curl and container IP:**
    - curl http://<container-ip>
7. **Run a container in detached mode, mapping port 80 on the host to port 80 in the container:**
    - docker run -d -p 80:80 <image-name or image-id>
8. **Run a container in detached mode with port mapping and a custom name (vishal):**
    - docker run -d -p 80:80 --name vishal <image-name or image-id>
9. **Remove unused images:**
    - docker image prune -f
10. **Remove all stopped containers:**
    - docker container prune -f

1. **Delete a specific running container:**
    - docker stop <container-id> && docker rm <container-id>
2. **Stop container:**
    - docker stop <container_name_or_id> # or
    - docker container stop <container_name_or_id> # or
    - docker stop $(docker ps -aq) # all container stop
3. **After stopping a container with docker stop, you can restart it using:**
    - docker start <container_name_or_id> # or
    - docker container start <container_name_or_id> # or
    - docker start $(docker ps -aq) # all container start
4. **Remove Container:**
    - docker rm <container_name_or_id> # or
    - docker container rm <container_name_or_id> #or
    - docker rm $(docker ps -aq) # all stopped container remove
5. **Remove Image:**
    - docker image rm <image-id or image-name>
    - docker rmi <image-id or image-name>
    - docker rmi $(docker images -q) # all image are remove
6. **Create image for container:**
    - docker commit <container-id> <image-name>
7. **Saves the entire Docker image (including layers, metadata, and history):**
    - docker save image-id>file.tar
8. **Exports the container's filesystem only (not the image layers or metadata):**
    - docker export container-id > file.tar

Docker Volume

1. Bind Volume

- A bind volume maps a directory or file on the host system to a directory in the container.
- The exact directory on the host is specified by the user.
- Use Case: Ideal for scenarios where you need to share files between the host and the container or when you want to debug by accessing the container files directly.
- **Example-**
  - mkdir mydata
  - cd mydata
  - nano index.html # Create html page
  - docker run -d -v /home/ubuntu/mydata:/usr/share/nginx/html nginx

2. Named Volume

- A named volume is managed by Docker and has a specific name given by the user.
- Docker decides where to store the data on the host machine.
- Volumes persist even after the container is deleted, making them great for long-term storage.
- Use Case: Suitable for data that needs to persist across container lifecycles but does not need direct access from the host.
- cd /var/lib/docker/volume # Named volume path
- **Example-**
  - docker volume create my-volume
  - docker run -v my-volume:/container/path my-image
  - docker volume ls # list of volume show
  - docker volume rm ls # remove volume
  - docker volume prune # remove unused volume

3. Anonymous

- An anonymous volume is created by Docker without a specific name.
- They are automatically deleted when the container is removed, unless explicitly retained.
- Use Case: Useful for temporary data storage that only needs to exist during the container's lifecycle.
- **Example-** docker run -v /container/path my-image

Wordpress

1. **Create mysql container:**

   docker run -d –name mywpdb -e MYSQL_ROOT_PASSWORD= Pass123 -e MYSQL_DATABASE=wordpress mysql

2. **Create WordPress & connect to mysql container:**

   docker run -d -p 80:80 –name mywordpress -e WORDPRESS_DB_HOST= mywpdb -e WORDPRESS_DB_USER=root -e WORDPRESS_DB_PASSWORD= Pass123 -e WORDPRESS_DB_NAME= wordpressdb --link mywpdb (# database container) =mywordpress (# wordpress container) wordpress

Docker file

1. Create a own file:

nano Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install nginx -y
EXPOSE 80
WORKDIR /var/www/html/
RUN touch index.html
RUN echo "Hello from Dockerfile" > index.html
CMD ["nginx", "-g", "daemon off;"]
```

2. Create Image Using Dockerfile:

docker build . # only run Dockerfile name of own file
docker build -f docker-file-name . # Run docker-file name not a Dockerfile
docker build -t myimg -f docker-file . # (-t = add name to image)

3. Attributes of Dockerfile:

1. FROM - use for base image or to pull image (multiple time use)
2. RUN - for install any software or for all Linux command (multiple time use)
3. EXPOSE - to open port no
4. COPY - to copy file & directory from host to images
5. ENV - to set environment variables
6. CMD - specifier the command to RUN when a container is run from images (use only one times)
7. ENTRYPOINT- specifies the command to run when a container is run from images but allow additional argument to be passed in (use only one times)
8. ADD - Copies files from host to images download zip or tar files from given link & extract it auto.
9. ARG - Define variable that passed to container while building images
10. VOLUME - create volume, to set volume
11. WORKDIR - to set working directory
12. MAINTAINER - to set name & email of author/ user
13. LABEL - To add metadata (data about data)
14. USER - To set user (root, ec2-user, docker etc)
15. HEALTHCHECK - to specifies path for health check or check health of mentioned url
16. SHELL-specifies shell to be used to run command
17. STOPSIGNAL - Specifies the signed to sent to container want to stop container gracefully
18. ONBUILD - Specifies the instruction to be used when we uses this images as base images for another images

Python

1. **Python framework –**
   - Django
   - Flask
2. **Requirements-** # we use Flask
   - Dockerfile
   - Requirement.txt
   - App.py
3. **Example-**
   - mkdir flask-app # create directory
   - cd
   - nano Dockerfile

```
FROM python:3.9-slim-buster
WORKDIR /app
COPY ./requirements.txt .
RUN pip install --upgrade pip && pip install -r requirements.txt
COPY . .
EXPOSE 5000
ENV FLASK_APP=app.py
CMD ["flask", "run", "--host=0.0.0.0"]
```

   - nano requirements.txt

```
Flask==2.2.2
Jinja2==3.0.3
MarkupSafe==2.1.2
Werkzeug==2.2.3
gunicorn==20.1.0
```

   - nano app.py

```
from flask import Flask, jsonify
app = Flask(__name__)
# Root route
@app.route("/", methods=["GET"])
def home():
    return jsonify({"msg": "Welcome to the Flask App!"})
# /hello route
@app.route("/hello", methods=["GET"])
def say_hello():
    return jsonify({"msg": "Hello from Flask!"})
if __name__ == "__main__":
    # Debug should be disabled in production
    app.run(host="0.0.0.0", port=5000, debug=True)
```

   - docker build -t flask-app .
   - docker run -d -p 5000:5000 –name python-app flask-app

Node js

1. **Requirements-**
   - Package.json
   - Index.js
   - Dockerfile
2. **Example-**
   - mkdir node
   - cd
   - nano package. Json

```
{
   "name": "node-app",
   "description": "hello jenkins test app",
   "version": "0.0.1",
   "private": true,
"main":"index.js",
       "scripts":{
       "start":"node index.js"
},
   "dependencies": {
      "express": "3.12.0"
   },
   "devDependencies": {
      "mocha": "10.2.0",
      "supertest": "6.3.3"
   }
  }
```

   - nano index.js

```
var express = require('express');
var app = express();//Respond with "hello world" for requests that hit our
root "/"
app.get('/', function (req, res) {
 res.send('my Node.js file ');
});//listen to port 3000 by default
app.listen(process.env.PORT || 3000);
module.exports = app;
```

   - nano Dockerfile

```
# Use the official Node.js 16 base image
FROM node:16-slim
# Set the working directory inside the container
WORKDIR /usr/src/app
# Copy the package.json and package-lock.json files (if available)
COPY package.json ./
# Install dependencies
RUN npm install
# Copy the application code to the container
COPY . .
# Expose the application port
EXPOSE 3000
# Define the command to run your app
CMD ["npm", "start"]
```

   - docker build -t node-app
   - docker run -d -p 3000:3000 --name node-file node-app

Mysql & Wordpress

MySQL:

1. **mkdir mysql # create mysql directory**
2. **cd mysql**
3. **nano Dockerfile**

```
FROM mysql:8.0

ENV MYSQL_ROOT_PASSWORD=pass123
ENV MYSQL_DATABASE=wordpressdb

EXPOSE 3306
CMD ["mysqld"]
```

4. **docker build -t mysql-file .**
5. **docker run -d --name mydb mysql-file**

WordPress:

1. **mkdir wordpress** # create mysql directory
2. **cd wordpress**
3. **nano Dockerfile**

```
# Use the official WordPress image from Docker Hub
FROM wordpress:latest

# Set environment variables for WordPress to connect to MySQL
ENV WORDPRESS_DB_HOST=mydb:3306
ENV WORDPRESS_DB_NAME=wordpressdb
ENV WORDPRESS_DB_USER=root
ENV WORDPRESS_DB_PASSWORD=pass123

# Expose the WordPress port
EXPOSE 80
```

4. **docker build -t wordpress-file .**
5. **docker run -d -p 8080:80 --name mywp --link mydb: mywp wordpress-file**

Shellscript using Dockerfile

1. mkdir myshell # create myshell directory
2. cd myshell
3. nano myshell.sh

```
#!/bin/bash
mysql -u root -p pass123 <<EOF
create database facebook;
use facebook;
create table user(id int not null primary key auto_increment, name
varchar(100), address varchar(50));
insert into user values(1,"vishal","Pune");
EOF
```

4. nano mydockerfile

```
FROM mysql
MAINTAINER vishal
ENV MYSQL_ROOT_PASSWORD pass123
EXPOSE 3306
COPY myfile.sh /docker-entrypoint-initdb.d/myfile.sh
RUN chmod +x docker-entrypoint-initdb.d/myfile.sh
CMD ["mysqld"]
```

5. docker build -t mysql-file -f mydockerfile .
6. docker run -d --name cont1 -e MYSQL_ROOT_PASSWORD=pass123 mysql-file

Docker Network

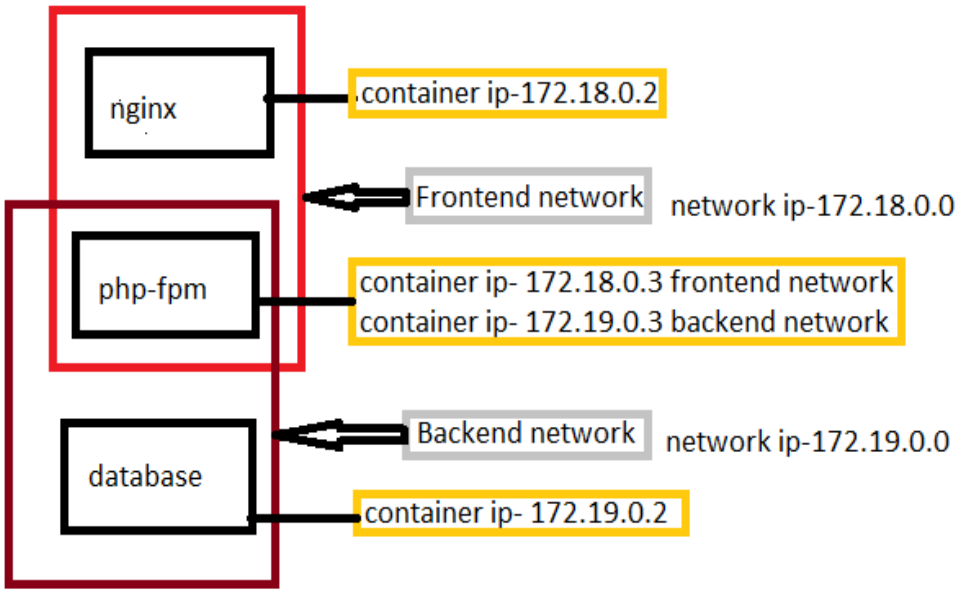
Type of network:

1. **Bridge Network (**default network):
    - **Description:** The default network mode in Docker, where containers connect to a bridge created on the host. Containers can communicate with each other using their IP addresses, and the host can also access them.
    - **Use Case:** Useful for standalone containers that need to communicate with each other on the same host.
    - **Example:** Containers running microservices on the same machine.
2. **Host Network** (insecure):
    - **Description:** In this mode, the container shares the host's network stack. It uses the host's IP address and ports directly, bypassing network isolation.
    - **Use Case:** Useful when performance is critical, or when the containerized app needs full network access without NAT.
    - **Example:** Network-intensive apps like monitoring tools or proxies.
3. **Overlays network** (communicate other networks):
    - **Description:** Allows containers running on different hosts to communicate securely over a distributed network. It uses an encrypted tunnel for inter-host communication.
    - **Use Case:** Ideal for multi-host Docker Swarm or Kubernetes environments.
    - **Example:** Services in a Docker Swarm or Kubernetes cluster communicating across nodes.

Commands:

1. **List Docker Networks:**
    - **docker network ls**
2. **Create a Docker Network:**
    - **docker network create <network_name>**
3. **Inspect a Docker Network:**
    - **docker network inspect <network_name>**
4. **Remove a Docker Network:**
    - **docker network rm <network_name>**
5. **Connect a Container to a Network:**
    - **docker network connect**
6. **Disconnect a Container from a Network:**
    - **docker network disconnect <network_name> <container_name_or_id>**

Example: Bridge network



1. **Create network:**
   - docker network create frontend
   - docker network create backend
2. **Create images:**
   - nano nginxdockerfile

```
FROM nginx
WORKDIR /usr/share/nginx/html
EXPOSE 80
RUN touch index.html
RUN echo "hello from nginx container" > index.html
CMD ["nginx", "-g", "daemon off;"]
```

   - nano phpdockerfile

```
FROM php:8.2-fpm
RUN apt-get update && apt-get install -y \
        libfreetype-dev \
        libjpeg62-turbo-dev \
        libpng-dev \
&& docker-php-ext-configure gd --with-freetype --with-jpeg \
&& docker-php-ext-install -j$(nproc) gd
```

   - nano mysqldockerfile

```
FROM mysql
ENV MYSQL_ROOT_PASSWORD pass123
EXPOSE 3306
CMD ["mysqld"]
```

- docker build -t nginximg -f nginxdockerfile
- docker build -t phpimg -f phpdockerfile
- docker build -t dbimg -f mysqldockerfile

3. **Create Container:**
   - **For Nginx container:**
     docker run -d -p80:80 --name myngcont --network frontend nginximg
   - **For php container:**
     docker run -d -p9000 --name myphpcont --network frontend phpimg
   - **For MySQL container:**
     docker run -d -p3306 --name mysqlcont --network backend dbimg
   - **Connect php container:**
     docker network connect backend myphpcont
4. **Find ip:**
   - docker network inspect frontend


```
Containers": {
        "192f4c7fe1184b2b918a31a8a8b4129865ea6691b01b1b7d3d34dc7b49f316ac": {
          "Name": "myngcont",
          "EndpointID": "98733190af6e168fbbcf3c44b8256f74726c021d3a2d6816fe7b888c21dc1115",
          "MacAddress": "02:42:ac:12:00:02",
          "IPv4Address": "172.18.0.2/16",
          "IPv6Address": ""
        },
        "ce11dc844c88ddce7646a953e2a18444aefeb7912479b18b9c0f87c681c65c54": {
          "Name": "myphpcont",
          "EndpointID": "7183ca60e42d781214993bb151e86fe75b1758164ecdb43f1c5f037c3e26a843",
          "MacAddress": "02:42:ac:12:00:03",
          "IPv4Address": "172.18.0.3/16",
          "IPv6Address": ""
        }
    }
```


   - docker network inspect backend


```
"Containers": {
        "c1ecb8d1b1e1ebb1e189f15d74330bcbc3aaba5bd5bf5d6e63cd36947359fe9d": {
          "Name": "mydbcont",
          "EndpointID": "4c6150bd44963e6830fed34e2b0ec4d11214b86c14033896750cf80f97293fd9",
          "MacAddress": "02:42:ac:13:00:03",
          "IPv4Address": "172.19.0.2/16",
          "IPv6Address": ""
        },
        "ce11dc844c88ddce7646a953e2a18444aefeb7912479b18b9c0f87c681c65c54": {
          "Name": "myphpcont",
          "EndpointID": "e34e1f536463272b032a8ae5dda455c5e1b2a1252189e6bbc8743666be97614c",
          "MacAddress": "02:42:ac:13:00:02",
          "IPv4Address": "172.19.0.3/16",
          "IPv6Address": ""
        }
    }
```


5. **Connect container mydbcont:** docker exec -it mydbcont /bin/bash
   - **Login mysql:** mysql -u root -p
     Password:pass123

- **Create database :** Create database facebook;
- Exit to mysql and container

6. **Connect container myphpcont:** docker exec -it myphpcont /bin/bash
   - apt update
   - apt install mariadb-server -y
   - **curl myngcont and mydbcont:**
     - curl http:// 172.18.0.2 # show the myngcont index page
     - mysql -u root -p -h 172.19.0.2
       Password:pass123
     - show databases: # show the database for mydbcont i.e facebook

ProxyServer

1. **mkdir proxyserver**
2. **cd proxyserver**
3. **docker network create mynetwork**
4. **docker run -d --name cont1 --network mynetwork nginx**
    a. docker exec -it cont1 /bin/bash
    b. cd /usr/share/ nginx/html/
    c. ls
    d. apt update
    e. apt install nano
    f. nano index.html
    g. edit index.html file
    h. exit
5. **docker run -d --name cont2 --network mynetwork nginx**
    a. docker exec -it cont2 /bin/bash
    b. cd /usr/share/ nginx/html/
    c. ls
    d. apt update
    e. apt install nano
    f. nano index.html
    g. edit index.html file
    h. exit
6. **docker run -d -p 80:80 --name proxyserver --network mynetwork nginx**
    a. docker exec -it proxyserver /bin/bash
    b. cd /etc/nginx
    c. apt update
    d. apt install nano
    e. nano nginx.conf

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    upstream webserver{
        server cont1:80;
        server cont2:80;
    }  # add yellow line in this file
```

    f. cd conf.d
    g. nano default.conf

```
    location / {
        proxy_pass http://webserver; # add this line
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
```

    h. service nginx restart
    i. exit
7. **public ip hit on browser**

Compose File

1. **mkdir compose**
2. **cd compose**
3. **nano docker-compose.yml**

```
services:
 db:
  image: mysql
  environment:
   MYSQL_ROOT_PASSWORD: pass123
   MYSQL_DATABASE: mydb
  ports:
   - 3306
 wordpress:
  image: wordpress
  ports:
   - 80:80
  environment:
   WORDPRESS_DB_HOST: db
   WORDPRESS_DB_USER: root
   WORDPRESS_DB_PASSWORD: poass123
   WORDPRESS_DB_NAME: mydb
```

4. **apt update**
5. **apt install docker-compose -y**
6. **docker-compose up -d # run the compose file**
7. **docker-compose down # remove all container and network created using compose file**

Three tier using docker-compose file

1. **Presentation Layer (Frontend):** Contains the user interface, usually index.html in a web app.
2. **Application Layer (Backend):** Manages business logic, typically with index.php.
3. **Data Layer (Database):** Stores and retrieves data, often using MySQL or similar databases.
4. **Example**:

   Directory Structure:

```
three-tier/
├── docker-compose.yml
├── html/
│   └── index.html
├── nginx/
│   ├── Dockerfile
│   └── nginx.conf
├── php/
│   └── Dockerfile
├── app/
│   └── index.php
└── initdb/
    └── init.sql
```

mkdir three-tier

cd three-tier

nano docker-compose.yml

```
version: '3.8'
services:
  web:
    build: ./nginx
    container_name: nginx_server
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html
    depends_on:
      - php-app
    networks:
      - app-network
  php-app:
    build: ./php
    container_name: php_app
    volumes:
      - ./app:/var/www/html
    networks:
      - app-network
      - db-network
  db:
    image: mysql:latest
    container_name: mysql_db
    environment:
      MYSQL_ROOT_PASSWORD: pass123
      MYSQL_DATABASE: user_data
    volumes:
      - db_data:/var/lib/mysql              # Persistent data storage
      - ./initdb:/docker-entrypoint-initdb.d  # SQL initialization scripts
    ports:
      - "3306:3306"                         # Expose MySQL port
    networks:
      - db-network
```

```
volumes:
  db_data:
networks:
  app-network:
  db-network:
```

mkdir html

cd html

nano index.html

```
<body>
    <div class="form-container">
        <h2>User Information</h2>
        <form action="index.php" method="POST">
            <div class="form-group">
                <label for="name">Name:</label>
                <input type="text" id="name" name="name" required>
            </div>
            <div class="form-group">
                <label for="gmail">Gmail:</label>
                <input type="email" id="gmail" name="gmail" required>
            </div>
            <div class="form-group">
                <label for="age">Age:</label>
                <input type="number" id="age" name="age" min="1" max="120" required>
            </div>
            <div class="form-group">
                <input type="submit" value="Submit">
            </div>
        </form>
    </div>
</body>
```

mkdir nginx

cd nginx

nano Dockerfile

```
FROM nginx:latest

COPY ./nginx.conf /etc/nginx/nginx.conf
```

nano nginx.conf

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    sendfile on;
    keepalive_timeout 65;

   server {
        listen 80;
       server_name localhost;
```

```
        location / {
            root /usr/share/nginx/html;
            index index.html index.php;
        }

        location ~ \.php$ {
            fastcgi_pass php_app:9000;
            fastcgi_index index.php;
            include fastcgi_params;
            fastcgi_param SCRIPT_FILENAME /var/www/html$fastcgi_script_name;
        }
    }

}
```

cd ..

mkdir php

nano Dockerfile

```
FROM php:8.2-fpm

# Install necessary extensions
RUN docker-php-ext-install mysqli pdo pdo_mysql

# Set the working directory
WORKDIR /var/www/html
```

Cd ..

mkdir app

cd app

nano index.php

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

// Database connection details
$host = "mysql_db"; // Change to your database host if needed
$username = "root";  // Your database username
$password = "pass123";      // Your database password
$dbname = "user_data"; // Name of the database

// Connect to the database
$conn = new mysqli($host, $username, $password, $dbname);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = htmlspecialchars($_POST["name"]);
    $gmail = htmlspecialchars($_POST["gmail"]);
    $age = htmlspecialchars($_POST["age"]);

    // SQL query to insert data into the database
    $sql = "INSERT INTO users (name, gmail, age) VALUES (?, ?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("ssi", $name, $gmail, $age); // "ssi" - string, string, integer
```

```
    if ($stmt->execute()) {
        echo "<h1>Data Saved Successfully</h1>";
        echo "<p><strong>Name:</strong> $name</p>";
        echo "<p><strong>Gmail:</strong> $gmail</p>";
        echo "<p><strong>Age:</strong> $age</p>";
    } else {
        echo "<p>Error saving data: " . $conn->error . "</p>";
    }

    $stmt->close();
} else {
    echo "<p>No data submitted!</p>";
}

// Close the database connection
$conn->close();
?>
```

mkdir initdb

cd initdb

nano init.sql

```
    CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    gmail VARCHAR(255) NOT NULL,
    age INT NOT NULL
);
```

docker-compose up -d

Custom MySQL with Docker Compose

1. **mkdir myfile**
2. **cd myfile**
3. **nano Dockerfile**

```
FROM mysql:8.0
ENV MYSQL_ROOT_PASSWORD=pass123
ENV MYSQL_DATABASE=mydb
EXPOSE 3306
CMD ["mysqld"]
```

4. **nano docker-compose.yml**

```
services:
 db:
  build:
   context: .
   dockerfile: Dockerfile
  container_name: my_db
  volumes:
   - mydata:/var/lib/mysql
  networks:
   - backend
  environment:
   MYSQL_ROOT_PASSWORD: pass123
   MYSQL_DATABASE: mydatabase
  ports:
   - 3306
networks:
 backend:
volumes:
 mydata:
```

5. **docker-compose up -d**
6. **docker ps -a**
7. **docker exec -it**

Docker Portainer

1. **Portainer is a lightweight, open-source management tool for Docker and Kubernetes environments.**
2. **It provides a user-friendly web interface to manage containers, images, networks, and volumes, simplifying container orchestration tasks.**
3. **Portainer supports role-based access control, application deployment, and monitoring, making it ideal for both developers and system administrators to manage containerized applications efficiently.**
4. **Commands:**

    a. docker volume create portainer_data
    b. docker run -d -p 8000:8000 -p 9000:9000 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce