



Docker

Docker

- **What is Docker?**

Docker is Open-Source containerization platform that enables developers and organizations to build, deploy, and manage application in isolated environments, ensuring consistency, and efficiency across various computing environments.

- **Difference between Virtualization & Containerization?**

- Virtualization :

- It involves creating virtual machine (VM) that run a full operating system on top of physical machine.
 - How it works : A hypervisor runs on the physical hardware and allows multiple VM's to operate on single machine.
 - Isolation Level is strong each VM is completely isolated with its own OS and Resources.
 - Tools :- VMware, Hyper-V, VirtualBox.

- Containerization :

- It Shares the Operating System (OS).
 - It involves packaging an application and its dependencies into a container. Which runs on one operating system and provides isolation.
 - How it works : Containers share the host OS but isolate the application environment using kernel features like namespace and cgroups.
 - Isolation Level, Process-Level. Isolation not as strong as VM but sufficient.
 - Tools :- Docker, Podman, Containerd.

- **Why use Docker?**

- Consistency Across Environments: Ensures your app runs the same on dev, test, and production.
- Lightweight: Container share the host OS kernel.
- Portability: Once built, Docker Containers can run anywhere. (On-Premises, Cloud, or Hybrid environments)

- **Core Components of Docker.**

- Docker Engine - Runtime that builds and runs containers.
- Docker Images - Templates used to create containers.
- Docker Containers - Lightweight, executable packages of software.
- Dockerfile - Script with instruction to build a Docker image.
- Docker Hub - Cloud based registry service for shareing and storing Docker images.
- Docker Compose - Running multi-container Docker applications using YAML/DockerCompose files.

- **Advantages of Docker Over Virtual Machines**

Feature	Virtual Machines	Docker Containers
Boot Time	Minutes	Seconds
Resource Usage	Heavy (OS per VM)	Lightweight (shared OS)
Performance	Less efficient	Near-native
Portability	Limited (dependent on hypervisor)	High (runs anywhere Docker is supported)
Image Size	Large (GBs)	Small (MBs to low GBs)

Docker & Docker Compose Installation.

RHEL-based systems

- `sudo yum update -y / sudo dnf update -y`

- `sudo yum install docker -y`
- `sudo systemctl start docker`
- `sudo systemctl enable docker`

Debian-based systems

- `sudo apt update -y`
- `sudo apt install docker.io -y`
- `sudo systemctl start docker`
- `sudo systemctl enable docker`

Docker-Compose

- `sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`
- `sudo chmod +x /usr/local/bin/docker-compose`

Optional -

- `sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose`

Verify Installation -

- `docker --version`
 - `docker-compose --version`
-

Docker Commands

- `docker ps` / `docker container ls`

Purpose: Lists running containers.

Flags:

- `-a` : Show all containers (default shows only running container)
- `-q` : List only container IDs

`-s` : Shows filesystem size for each container

```
docker ps          # docker container ls ## both works same
docker ps -s
docker stop $(docker ps -aq) # this will stop all running containers
```

- `docker images` / `docker image ls`

Purpose: List local images.

Flags:

`-a` : Shows all images

`-q` : Only display image IDs

```
docker images
docker rmi $(docker images -aq) # this will remove/delete all the local images
```

- `docker search <img-name>`

Purpose: Search for images on Docker Hub.

Flags:

`--limit <number>` : Limits the numbers of results.

`--filter is-official=true` : Shows only official images.

```
docker search busybox
```

- `docker pull <img-name>`

Purpose: Download an image from Docker Hub.

```
docker pull nginx
```

- `docker create`

Purpose: Create a container from an image (but in stop state, use start command to get it running)

Flags:

`--name <container-name>` : Assign a name to the container.

`-p <host-port>:<container-port>` : Bind container port to a port on the host.

`-v <host>:<container>` : Volume mount.

```
docker create --name web httpd
```

- `docker start <container-id>`

Purpose: Start a stopped container.

```
docker start web
```

- `docker stop <container-id>`

Purpose: Stop a running container gracefully.

```
docker stop web
```

- `docker rm <container-id>`

Purpose: Remove one or more stopped containers.

Flag:

`-f`: Force removal of a running container.

```
docker rm web
```

- `docker rmi <image-id>`

Purpose: Remove one or more Docker images.

Flag:

`-f` : Force removal even if image is used by containers.

```
docker rmi httpd
```

- `docker run -d`

Purpose: Run a container from an image in detached mode (in the background).
Flags:

- `-d` : Run in detached mode.
- `--name <container-name>` : Assign a name to the container.
- `-p <host-port>:<container-port>` : Map Ports.
- `-v <host>:<container>` : Volume mount.
- `-e <key>:<value>` : Set environment variables in the container. If any.
- `--network <network-name>` : Connect the container to a user-defined Docker network.
- `--restart <policy>` : Restart policy. (no, on-failure, always, unless-stopped)
- `--env-file <file.env>` : Load environment variables from a file. (Create a .env file)
- `--cpus <value>` : Limit number of CPUs the container can use.
- `-m <value>` : Set a memory limit for the container (e.g., 512m, 1g).

```
docker run -d --name database -e MYSQL_ROOT_PASSWORD=12345 mysql
docker run -d --env-file sql.env --name database mysql
```

- `docker exec -it <container-id> bash`

Purpose: Run a command in a running container (interactively).
Flags:

- `-i` : Interactive Mode.
- `-t` : Allocate a pseudo-TTY/Terminal.

```
docker exec -it web /bin/bash
```

- `docker commit <container-id> <name-image>`

Purpose: Create a new image from a container.

```
docker commit web custom-nginx:v1
```

- `docker save -o <file.tar> <img-name:tag>`

Purpose: Save an image to a tar archive file.

Flag:

`-o <file.tar>` : Output filename.

```
docker save -o nginx.tar nginx:latest
```

- `docker load -i <file.tar>`

Purpose: Load an image from a tar archive.

Flag:

`-i` : Input file

```
docker load -i nginx.tar
```

- `docker history <image-id>`

Purpose: Show the history of an image.

```
docker history nginx
```

- `docker login -u <username>`

Purpose: Authenticate to a Docker registry.

```
docker login -u user1
```

- `docker tag <img-name:tag> <new-name:tag>`

Purpose: Tag an image for a specific registry or name.

```
docker tag nginx:latest myregistry.com/mynginx:prod
```

- `docker push`

Purpose: Push an image to a registry.

```
docker push myregistry.com/mynginx:prod
```

- `docker network`

Purpose: Manage Docker networks.

Sub-Commands:

`ls` : List docker networks.

`create` : Create network. (To define type use `—driver`)

`rm` : Remove created network.

`inspect` : To get more details of network.

`connect <network-name> <container-name>` : Connect network to container.

`disconnect <network-name> <container-name>` : Disconnect connected network of container.

```
docker network ls
```

```
docker network create frontend --driver bridge    # by default it create a bridge
```

- `docker volume`

Purpose: Manage Docker volumes.

Sub-Commands:

`ls` : List volumes.

`create` : Create a new volume.

`rm` : Remove a volume.

`inspect` : Inspect volume details.

```
docker volume ls
```

```
docker volume create my-volume1    # Named Volume
```

```
docker volume inspect my-volume1
```

```
docker volume rm my-volume1
```

- `docker system prune`

Purpose: Remove all unused containers, networks, images, and optionally volumes.

Flags:

`-a` : Remove all unused images.

`-f` : Do not prompt for confirmation. / Forcefully.

`--volumes` : Include volumes in cleanup.

```
docker system prune -a --volumes -f
```

- `docker container inspect <container-id>`

Purpose: Display detailed information on a container in JSON format.

```
docker container inspect web
```

- `docker stats <container-id>`

Purpose: Display a live stream of resource usage statistics (CPU, memory, network, I/O) for containers.

```
docker stats web
```

For More Docker Commands & Options

Official Docker Documentation -

<https://docs.docker.com/engine/reference/commandline/docker/>

Docker Network

Docker network, allows containers to communicate with —each other, host machine, or outside world (internet).

Types of Docker Networks :-

Driver	Description
bridge	The default network driver.
host	Remove network isolation between the container and the Docker host.

Driver	Description
none	Completely isolate a container from the host and other containers.
overlay	Overlay networks connect multiple Docker daemons together.
ipvlan	IPvlan networks provide full control over both IPv4 and IPv6 addressing.
macvlan	Assign a MAC address to a container.

Docker Volume

A Docker volume is a special type of storage used by Docker containers to store data permanently, even if the container is stopped, deleted, or recreated.

When you run a container, by default, any data it creates is lost when the container stops. Volumes solve this problem by keeping the data outside the container, in a separate location managed by Docker. This makes it easier to save, share, and back up important data.

Types of Docker Volumes :-

Named Volume

- Create and managed by docker
- Stored in Docker's default location (/var/lib/docker/volumes/)
- Can be reused across multiple containers.

```
docker volume create my-volume  
docker run -d -v my-volume:/app/data my-image
```

Anonymous Volumes

- Similar to named volumes.
- We don't name it explicitly. Docker gives it a random name.

- Harder to manage because you don't control the name. Less flexible.

```
docker run -d -v /app/data my-image
```

Bind Mounts

- Links a specific file or folder from host machine into the container.
- Not managed by Docker.

```
docker run -d -v /host/path:/container/path my-image
```

- [/var/lib/docker — Docker's Default Storage Directory](#)

This is the default data directory where Docker stores all of its persistent data :-

Images

Containers

Volumes

Networks

So essentially, everything Docker runs or builds is stored here, unless configured.

Dockerfile

A Dockerfile is a text file that contains instructions on how to build a Docker image. Each instruction in the Dockerfile creates a **layer** in the image.

Basic Structure of a Dockerfile

```
# Example Dockerfile
FROM nginx
WORKDIR /usr/share/nginx/html
```

```
COPY ./html .
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx","-g","daemon off;"]
```

Build Dockerfile

```
docker build -t nginx-build .
# if your dockerfile name isn't "Dockerfile"
docker build -f <DockerFile-Name> -t nginx-build .
```

Dockerfile Attributes/Instruction

Instruction	Description	Example
FROM	Sets the base image (must be the first instruction).	FROM ubuntu:22.04
LABEL	Adds metadata to the image (e.g., maintainer, version).	LABEL maintainer="@example.com"
ENV	Sets environment variables.	ENV NODE_ENV=production
WORKDIR	Sets the working directory inside the container.	WORKDIR /usr/src/app
COPY	Copies files from the host to the container.	COPY ./app
ADD	Similar to COPY, but also supports remote URLs and automatic unpacking of compressed files.	ADD archive.tar.gz /app/
RUN	Executes a command while building the image (e.g., installing packages).	RUN apt-get update && apt-get install

Instruction	Description	Example
CMD	Sets the default command to run when the container starts (can be overridden).	<code>CMD ["node", "app.js"]</code>
ENTRYPOINT	Sets the main executable (like CMD , but harder to override).	<code>ENTRYPOINT ["python3"]</code>
EXPOSE	Documents the port the container listens on (for reference only).	<code>EXPOSE 80</code>
VOLUME	Creates a mount point with a volume attached.	<code>VOLUME /data</code>
ARG	Defines build-time variables (used only during image build).	<code>ARG VERSION=1.0.0</code>
USER	Sets the user that will run inside the container.	<code>USER appuser</code>
HEALTHCHECK	Sets a command to check container health.	<code>HEALTHCHECK CMD curl --fail http://localhost</code>
SHELL	Changes the default shell used in RUN commands (mostly for Windows images).	<code>SHELL ["powershell", "-command"]</code>
ONBUILD	Triggers a command in child images (used in base images meant to be extended).	<code>ONBUILD RUN npm install</code>
STOPSIGNAL	Sets the system call signal that will be sent to the container to exit.	<code>STOPSIGNAL SIGTERM</code>

CMD vs ENTRYPOINT

CMD is default and can be overridden at runtime.

ENTRYPOINT is always executed, and arguments passed at runtime are appended to it

MySQL Dockerfile

```
FROM mysql:latest
LABEL maintainer="example@example.com"
# COPY init.sql /docker-entrypoint-initdb.d
EXPOSE 3306
CMD ["mysqld"]
```

ENV variables that can be used -

ENV MYSQL_ROOT_PASSWORD=12345

ENV MYSQL_DATABASE=Database-name

ENV MYSQL_USER=CreateUser

ENV MYSQL_PASSWORD>PasswordForUser

Initialization SQL Scripts

If you want to run custom `.sql` files at container startup:
Create `.sh` or `.sql` file and COPY it in Docker image.

Postgres Dockerfile

```
FROM postgres
ENV POSTGRES_USER=root
ENV POSTGRES_PASSWORD=123
ENV POSTGRES_DB=mydatabase
EXPOSE 5432
```

Build and run -

- `docker build -t data-base .`
- `docker run -d --name pdb data-base`
- `docker exec -it pdb bash` then `psql -U root -d mydatabase`

MongoDB Dockerfile

```
FROM mongo
ENV MONGO_INITDB_ROOT_USERNAME=root
ENV MONGO_INITDB_ROOT_PASSWORD=123
ENV MONGO_INITDB_DATABASE=info
EXPOSE 27017
```

Build and run -

- `docker build -t mongodb .`
- `docker run -d --name mdb mongodb`
- `docker exec -it mdb mongosh -u root -p`

WordPress Dockerfile

```
FROM mysql:latest
ENV MYSQL_ROOT_PASSWORD=123
ENV MYSQL_DATABASE=Database-name
EXPOSE 3306
CMD ["mysqld"]
```

```
FROM wordpress:latest
ENV WORDPRESS_DB_HOST=db
ENV WORDPRESS_DB_NAME=Database-name
ENV WORDPRESS_DB_USER=root
ENV WORDPRESS_DB_PASSWORD=123
EXPOSE 80
```

Build the files and run them -

- `docker build -f <docker-filename> -t <name-the-build> .`
- `docker run -d --name db <img-name>`

```
- docker run -d -p <port-no>:80 --name <container-name-you-want> --link <container-name>:<container-img>
<img-name>
```

Node.js Dockerfile

File Structure

```
node-docker-app/
├── app.js          # Node.js app
├── package.json    # Dependencies and scripts
└── Dockerfile      # Docker build instructions
```

app.js

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello from Dockerized Node.js App!\n');
});

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

package.json

```
{
  "name": "node-docker-app",
  "version": "1.0.0",
  "description": "A simple Node.js app running inside Docker",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "author": "dunoo",
```



```
"license": "ISC"
}
```

Dockerfile

```
FROM node
WORKDIR /usr/src/app
COPY package.json .
RUN npm install
COPY app.js .
EXPOSE 3000
```

Build and run -

- `docker build -t nodejs .`
- `docker run -d -p <port-you-want>:3000 --name node-app nodejs`

Python (Flask) Dockerfile

File Structure

```
flask-docker-app/
├── app.py           # Flask application
├── requirements.txt # Python dependencies
└── Dockerfile       # Docker instructions
```

requirements.txt

```
flask
```

app.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
```

```
def home():
    return "Hello from Dockerized Flask App!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Dockerfile

```
FROM python
WORKDIR /app
COPY requirement.txt .
RUN pip install --no-cache-dir -r requirement.txt
COPY app.py .
EXPOSE 5000
CMD ["python","app.py"]
```

Build and run -

- `docker build -t py-app .`
- `docker run -d -p 5000:5000 --name py py-app`

Docker-Compose

A tool to define and run multi-container Docker applications. Docker compose file are written in YAML (Yet Another Markup Language) extension. (i.e., `.yaml` or `.yml`)

Format for file name, `docker-compose.yml` to configure app's services.

To run:

```
docker-compose up -d
```

Docker Compose Commands

<code>docker-compose --version</code>	# Displays docker-compose version
<code>docker-compose up -d</code>	# Start in detached mode
<code>docker-compose down</code>	# Stop and remove containers, networks, vc

```

docker-compose -f composefile.yml up -d # for file with different name
docker-compose config # validates docker-compose.yml file
docker-compose -f composefile.yml config # same as docker-compose config
docker-compose restart # Restart containers
docker-compose logs # View logs of all services
docker-compose up -d --build # Rebuilds images & starts containers in b

```

Basic docker-compose.yml Structure

```

version: '3.9'

services:
  service_name:
    image: or build:
    ports:
    environment:
    volumes:
    depends_on:
    command:
    networks:
    container_name:
networks:
volumes:

```

Directive	Example / Description
version	Compose file format version
services	Defines all containers
image	Use a prebuilt image
build	Build from a Dockerfile
ports	Expose container port to host ("8080:80")
volumes	Map host or named volumes to containers ("./data:/app/data")
environment	Set environment variables (- MYSQL_ROOT_PASSWORD=12345) / (MYSQL_ROOT_PASSWORD: "12345")

Directive	Example / Description
<code>command</code>	Override default command (like <code>CMD</code> in Dockerfile)
<code>depends_on</code>	Specify dependency order (e.g., wait for db before starting app)
<code>networks</code>	Define shared communication networks for services
<code>restart</code>	Auto-restart policy: <code>no</code> , <code>always</code> , <code>unless-stopped</code> , <code>on-failure</code>

- **3 Tier Architecture**

- My Directory Structure

```

/3arch/
├── app
│   ├── code
│   │   ├── test.php
│   │   └── submit.php
├── db
│   ├── Dockerfile
│   └── init.sql
├── docker-compose.yml
└── web
    ├── code
    │   ├── form.html
    │   └── index.html
    ├── config
    └── default.conf

```

- docker-compose.yml

```

services:
  db:
    build:
      context: /3arch/db/
      dockerfile: Dockerfile
    container_name: db
    volumes:
      - myvolume:/var/lib/mysql/

```

```
networks:
  - dbnet
healthcheck:
  test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
  interval: 10s
  timeout: 5s
  retries: 5
app:
  image: bitnami/php-fpm
  container_name: app
  volumes:
    - /3arch/app/code:/app/
  networks:
    - dbnet
    - webnet
  depends_on:
    - db
web:
  image: nginx
  ports:
    - "80:80"
  container_name: web
  volumes:
    - /3arch/web/code:/usr/share/nginx/html/
    - /3arch/web/config:/etc/nginx/conf.d/
  networks:
    - webnet
  depends_on:
    - db
    - app
networks:
  webnet: {}
  dbnet: {}
volumes:
  myvolume: {}
```

- Dockerfile

```
FROM mysql
ENV MYSQL_ROOT_PASSWORD=Pass1972
ENV MYSQL_DATABASE=info
COPY init.sql /docker-entrypoint-initdb.d
EXPOSE 3306
CMD ["mysqld"]
```

- init.sql

```
use info;
create table users(id int primary key auto_increment, name varchar(20), ema
```

- default.conf

```
server {
    listen    80;
    listen  [::]:80;
    server_name localhost;

    #access_log /var/log/nginx/host.access.log main;

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

    #error_page 404          /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
```

```

}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80
#
#location ~ /\.php$ {
#    proxy_pass http://172.21.0.2;
#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
location ~ /\.php$ {
    root            /app;                                # root html; → Defines the doc
                                                         # fastcgi_pass 127.0.0.1:9000; → S
                                                         # fastcgi_index index.php; → If a c
    fastcgi_pass    app:9000;                            # $fastcgi_script_name
    fastcgi_index   index.php;                            # Example of How This
    fastcgi_param   SCRIPT_FILENAME /app/$fastcgi_script_name; # http
                                                         # $fastcgi_script_name will be /inc
                                                         # So /scripts$fastcgi_script_name
                                                         # /scripts/index.php

    include         fastcgi_params;
}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny all;
#}
}

```

- submit.php

```

<?php
$name=$_POST["name"];
$email=$_POST["email"];
$website= $_POST["website"];
$comment=$_POST["comment"];
$gender=$_POST["gender"];

echo $name;
echo $email;
echo $gender;
echo $comment;
echo $website;

$servername = "db";
$username = "root";
$password = "";
$dbname = "info";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO users (name,email,website,comment,gender)VALUES('";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

```



```
mysqli_close($conn);  
?>
```

- o form.html

```
<!DOCTYPE HTML>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Form</title>  
  <style>  
    /* Basic Reset */  
    * {  
      margin: 0;  
      padding: 0;  
      box-sizing: border-box;  
      font-family: 'San Francisco', -apple-system, BlinkMacSystemFont, 'Se  
    }  
  
    body {  
      display: flex;  
      justify-content: center;  
      align-items: center;  
      height: 100vh;  
      background: linear-gradient(135deg, #000, #1b1b1b);  
      color: white;  
      padding: 20px;  
    }  
  
    /* Glassmorphism Effect */  
    .form-container {  
      background: rgba(255, 255, 255, 0.05);  
      backdrop-filter: blur(15px);  
      padding: 40px;
```

```

border-radius: 20px;
width: 100%;
max-width: 450px;
box-shadow: 0 4px 30px rgba(0, 0, 0, 0.5);
text-align: center;
animation: fadeIn 1s ease-in-out;
}

/* Title */
h2 {
font-size: 28px;
font-weight: 600;
margin-bottom: 20px;
color: #fff;
text-transform: uppercase;
letter-spacing: 1.5px;
opacity: 0;
animation: fadeIn 1.2s ease-in-out forwards;
}

/* Form Input Fields */
input, textarea {
width: 100%;
padding: 12px;
margin: 10px 0;
border-radius: 8px;
border: none;
background: rgba(255, 255, 255, 0.15);
backdrop-filter: blur(10px);
color: white;
font-size: 16px;
transition: all 0.3s ease;
outline: none;
opacity: 0;
animation: fadeInUp 1s ease-in-out forwards;
}

```

```

textarea {
    resize: none;
    height: 100px;
}

/* Gender Selection */
.gender-options {
    display: flex;
    justify-content: space-between;
    margin: 15px 0;
    opacity: 0;
    animation: fadeInUp 1.2s ease-in-out forwards;
}

.gender-options label {
    display: flex;
    align-items: center;
    cursor: pointer;
}

.gender-options input {
    margin-right: 10px;
    transform: scale(1.2);
}

/* Submit Button */
input[type="submit"] {
    width: 100%;
    padding: 15px;
    border: none;
    border-radius: 50px;
    font-size: 18px;
    font-weight: 600;
    cursor: pointer;
    background: linear-gradient(145deg, #1e1e1e, #282828);
}

```

```

    box-shadow: 5px 5px 15px #0a0a0a, -5px -5px 15px #333;
    color: #fff;
    transition: all 0.3s ease;
    opacity: 0;
    animation: fadeInUp 1.4s ease-in-out forwards;
}

input[type="submit"]:hover {
    background: linear-gradient(145deg, #282828, #1e1e1e);
    box-shadow: 3px 3px 10px #000, -3px -3px 10px #3a3a3a;
    transform: scale(1.02);
}

/* Animations */
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(-10px); }
    to { opacity: 1; transform: translateY(0); }
}

@keyframes fadeInUp {
    from { opacity: 0; transform: translateY(20px); }
    to { opacity: 1; transform: translateY(0); }
}
</style>
</head>
<body>

<div class="form-container">
    <h2>Form</h2>
    <form method="post" action="submit.php">
        <input type="text" name="name" id="name" placeholder="Your Name"
        <input type="email" name="email" id="email" placeholder="Your Email"
        <input type="text" name="website" id="website" placeholder="Your We
        <textarea name="comment" id="comment" placeholder="Your Feedback

        <div class="gender-options">

```

```

    <label><input type="radio" name="gender" value="female" required:
    <label><input type="radio" name="gender" value="male" required> |
    <label><input type="radio" name="gender" value="other" required>
  </div>

  <input type="submit" name="submit" value="Submit">
</form>
</div>

</body>
</html>

```

- **Flask App + MySQL**

- My Directory Structure

```

flask-sql/
├── app/
│   ├── app.py
│   ├── requirements.txt
│   └── Dockerfile
├── docker-compose.yml
└── init.sql

```

- app.py

```

from flask import Flask, jsonify
import mysql.connector

app = Flask(__name__)

@app.route('/')
def hello():
    try:
        conn = mysql.connector.connect(
            host='db-sql',
            user='root',

```

```

        password='123',
        database='myapp'
    )
    cursor = conn.cursor()
    cursor.execute("SELECT message FROM greetings LIMIT 1;")
    result = cursor.fetchone()
    return jsonify({"message": result[0]})
except Exception as e:
    return jsonify({"error": str(e)})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

- requirement.txt

```

flask
mysql-connector-python

```

- Dockerfile

```

FROM python
WORKDIR /app
COPY requirement.txt .
RUN pip install --no-cache-dir -r requirement.txt
COPY app.py .
EXPOSE 5000
CMD ["python","app.py"]

```

- docker-compose.yml

```

services:
  flask-app:
    build:
      context: ./app
      dockerfile: Dockerfile

```

```

    container_name: flask-app
    depends_on:
      - db-sql
    networks:
      - appnet
    ports:
      - "5000:5000"
  db-sql:
    image: mysql
    ports:
      - "3306"
    container_name: db-sql
    environment:
      - MYSQL_ROOT_PASSWORD=123
      - MYSQL_DATABASE=myapp
    networks:
      - appnet
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
      - volume1sql:/var/lib/mysql
  networks:
    appnet: {}
  volumes:
    volume1sql: {}

```

- init.sql

```

CREATE DATABASE IF NOT EXISTS myapp;
USE myapp;
CREATE TABLE greetings (
  id INT AUTO_INCREMENT PRIMARY KEY,
  message VARCHAR(250) NOT NULL
);
INSERT INTO greetings (message) VALUES ('Hello from Flask + MySQL!');

```

- **Node.js + MongoDB**

- My Directory Structure

```
nodejs-mongodb
├── app
│   ├── Dockerfile
│   ├── app.js
│   └── package.json
└── docker-compose.yml
```

- Dockerfile

```
FROM node
WORKDIR /app
COPY package.json .
RUN npm install
COPY app.js .
EXPOSE 3000
CMD ["npm","start"]
```

- app.js

```
const express = require('express');
const mongoose = require('mongoose');

const app = express();
const port = 3000;

// MongoDB connection      password-and-host-and-user
mongoose.connect('mongodb://root:123@mongo-db:27017/mydb?authSource=admin', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
```



```

db.once('open', () => console.log('Connected to MongoDB'));

// Define a sample schema and model
const UserSchema = new mongoose.Schema({ name: String });
const User = mongoose.model('User', UserSchema);

// Route
app.get('/', async (req, res) => {
  const user = await User.create({ name: 'Luffy' });
  res.json({ message: 'User added!', user });
});

app.listen(port, () => {
  console.log(`App listening on port ${port}`);
});

```

- package.json

```

{
  "name": "node-mongo-app",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "^4.18.0",
    "mongoose": "^7.0.0"
  }
}

```

- docker-compose.yml

```

services:
  mongo-db:

```

```
image: mongo
ports:
  - "27017:27017"
container_name: mongo-db
environment:
  - MONGO_INITDB_ROOT_PASSWORD=123
  - MONGO_INITDB_ROOT_USERNAME=root
networks:
  - node-mongo
volumes:
  - mongodb:/data/db
node-js:
  build:
    context: ./app
    dockerfile: Dockerfile
  container_name: node-js
  ports:
    - "80:3000"
  networks:
    - node-mongo
  depends_on:
    - mongo-db
  volumes:
    mongodb: {}
  networks:
    node-mongo: {}
```

Portainer

Portainer is a lightweight management UI that allows to **easily** manage Docker environments (containers, images, networks, volumes, etc.) via a web interface.

```
docker volume create data
docker run -d -v /var/run/docker.sock:/var/run/docker.sock -v data:/data -p 9000
```

```
# Runs Portainer (a Docker GUI) in detached mode:  
# - Mounts Docker socket to manage Docker.  
# - Stores data in data volume.  
# - Exposes web UI on ports 9000 (HTTP) and 9443 (HTTPS).  
# - Names the container portainer-ui.
```