DOCKER

Docker is an open platform for developing , shipping and running applictaions. OR It is a platform which packages an applictaion and all its dependencies together in the form of containers.

Why we need docker ?

Developers often face issues where an app works on their computer but breaks elsewhere (e.g., production).

Why it happens: Different OS, Python/Java/C# versions, missing packages, etc.

Docker solution:

Package everything (code + dependencies + OS environment) into a container that runs the same everywhere — on your laptop, in staging, or in the cloud.

Docker file , Docker image and Docker conatiner

1. Dockerfile

A Dockerfile is a text file with instructions on how to build a Docker image i.e. text document which contains all command that a user can call on the command line to assemble/create an image

2. Docker Image

A Docker Image is the result of building a Dockerfile. Template to create a docker container. It's a snapshot of an application with all its dependencies. It is the blueprint of creating docker container

3. Docker Container

A Docker Container is a running instance of a Docker Image.It hold entire package to run the applictaion. It’s isolated, lightweight, and can be started, stopped, or deleted.

Think of it like running the app from the image.

Dockerizing our app

Example: Simple Python App

Step 1: Dockerfile

Create a file named Dockerfile:

```
# Use official Python image
FROM python:3.9-slim

# Set working directory
WORKDIR /app

# Copy files
COPY . .

# Install dependencies
RUN pip install -r requirements.txt

# Run the app
CMD ["python", "app.py"]
```

DOCKER FILE COMMAND

Summary Table

Instruction	Purpose	When It Runs
`FROM`	Sets base image	At the start of build
`WORKDIR`	Sets working directory	Affects all following lines
`COPY`	Copies files into image	During build
`RUN`	Runs command while building image	During build
`CMD`	Default command for container	When container runs
`EXPOSE`	Documents exposed ports	During build (docs only)
`ENV`	Sets env vars inside container	During build + runtime

Step 2: Project Structure

```
demo/
├── app.py
├── requirements.txt
└── Dockerfile
```

app.py

```
print("Hello from Docker!")
```

requirements.txt

```
# no dependencies for this example
```

⚙️ Step 3: Build the Docker Image

Move to the demo directory

```
Run this in the terminal:
docker build -t demo .
```

🧱 Detailed Breakdown

Part	Meaning
`docker`	The Docker command-line tool
`build`	Tells Docker to build an image
`-t demo`	Tags the image with the name `demo`
`.`	Tells Docker to look for the `Dockerfile` in the current directory

🖼️ What happens when you run it?

Docker:

1. **Looks for a file named** `Dockerfile` in your current directory (`.`).
2. **Reads the instructions** from it (e.g., base image, files to copy, install commands).
3. **Executes those instructions** to create a Docker image.
4. **Saves the image locally** with the name (or "tag") `demo` .

🚀 Step 4: Run the Docker Container

```
docker run demo
```

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
demo         latest    18935e99e185   8 minutes ago   132MB
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker run demo
Hello from Docker!
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo>
```

Docker command

-it stand for interactive mode i.e. we can acces the terminal of ruuning container (example: docker run -it ubuntu)

`1) version ->` docker -v OR docker --version

`2) pull image from docker hub ->` docker pull [image-name]

1. **Pull** a Docker image
2. **Run** a container from it
3. **Check** if it's running or stopped

`docker pull mysql`	Pulls the **latest** version of the official MySQL image from Docker Hub. This is typically the `latest` tag.
`docker pull mysql:8.0`	Pulls a **specific version**, i.e., **MySQL 8.0** image.

`3) run the image ->` docker run [image-name]

`4) searches Docker Hub ->` docker search mysql (It searches Docker Hub (the default public image registry) for images related to mysql.)

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker search mysql
NAME                 DESCRIPTION                                     STARS     OFFICIAL
mysql                MySQL is a widely used, open-source relation…   15871     [OK]
bitnami/mysql        Bitnami container image for MySQL               140
circleci/mysql       MySQL is a widely used, open-source relation…   32
bitnamicharts/mysql  Bitnami Helm chart for MySQL                    0
cimg/mysql                                                           3
ubuntu/mysql         MySQL open source fast, stable, multi-thread…   69
google/mysql         MySQL server for Google Compute Engine          26
linuxserver/mysql    A Mysql container, brought to you by LinuxSe…   46
```

`5) lists all Docker containers on your system — both Running and Stopped ->` docker ps -a

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps -a
CONTAINER ID   IMAGE          COMMAND          CREATED             STATUS                         PORTS     NAMES
07467a270267   hello-world    "/hello"         13 minutes ago      Exited (0) 13 minutes ago                awesome_bhaskara
ef8ce56f9ca9   qwerty:latest  "python app.py"  About an hour ago   Exited (0) About an hour ago             silly_williamson
c1ad5021108e   qwerty         "python app.py"  About an hour ago   Exited (0) About an hour ago             naughty_wu
```

`6) rename my image ->` docker run --name tester -d [Image name / container Id]

`docker run`	Run a new container
`--name tester`	Name the container `tester` (instead of a random name)
`-d`	Run it in **detached mode** (in the background)
`qwerty`	Use the Docker image named `qwerty`

`7. to run container in interactive modec->` docker run --name tester(new image name) -it -d qwerty(image name)

- `-it` is used for **interactive terminals** (like bash shells)
- `-d` is used to **detach** the container and run it in the background

8) check only container which are running -> docker ps

9) To enter in docker container -> docker exec -it [Container Id] [COMMAND]

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps -a
CONTAINER ID   IMAGE          COMMAND           CREATED              STATUS                          PORTS     NAMES
42626c622491   python         "python3"         About a minute ago   Exited (0) About a minute ago             admiring_bell
b9473ec7843e   qwerty         "python app.py"   12 minutes ago       Exited (0) 12 minutes ago                 newtester
998961e9b412   qwerty         "python app.py"   19 minutes ago       Exited (0) 19 minutes ago                 tester
07467a270267   hello-world    "/hello"          37 minutes ago       Exited (0) 37 minutes ago                 awesome_bhaskara
ef8ce56f9ca9   qwerty:latest  "python app.py"   2 hours ago          Exited (0) 2 hours ago                    silly_williamson
c1ad5021108e   qwerty         "python app.py"   2 hours ago          Exited (0) 2 hours ago                    naughty_wu
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker run --name sundari -it -d pyhton
Unable to find image 'pyhton:latest' locally
docker: Error response from daemon: pull access denied for pyhton, repository does not exist or may require 'docker login': denied: requested access to the resource is denied

Run 'docker run --help' for more information
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker run --name sundari -it -d python
9715289d1d92aef715922743077707f769b1259ac8136e342ce230dd55fedf17
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
9715289d1d92   python    "python3"   12 seconds ago   Up 11 seconds             sundari
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> ^C
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker exec -it 9715289d1d92 python3
Python 3.13.5 (main, Jul 22 2025, 04:26:21) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello -world")
hello -world
>>> print(3+5)
8
>>>
```

✅ Summary of Commands

Purpose	Command
Pull image	`docker pull python`
Run container	`docker run --name mypython -it python`
Show running	`docker ps`
Show all (incl. exited)	`docker ps -a`

10) Inspect returns detailed, low-level information about a Docker container or image. -> docker inspect 9715289d1d92aef71592274307

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker inspect 9715289d1d92aef715922743077707f769b1259ac8136e342ce230dd55fedf17
[
    {
        "Id": "9715289d1d92aef715922743077707f769b1259ac8136e342ce230dd55fedf17",
        "Created": "2025-07-30T08:49:54.905681179Z",
        "Path": "python3",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1866,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-07-30T08:49:54.951431234Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:4bbf6d52fe8de40e9e20b1453e14abb40eb1c73cd1d2b22e8ed015ad0817a088",
        "ResolvConfPath": "/var/lib/docker/containers/9715289d1d92aef715922743077707f769b1259ac8136e342ce230dd55fedf17/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/9715289d1d92aef715922743077707f769b1259ac8136e342ce230dd55fedf17/hostname",
```

11) Remove a running container -> docker rm [container-id]

🔴 Option 1: Stop the container, then remove it

bash Copy Edit

```
docker stop 9715289d1d92
docker rm 9715289d1d92
```

💥 Option 2: Force remove (kills it immediately)

bash Copy Edit

```
docker rm -f 9715289d1d92
```

> ⚠️ Only use `-f` if you're okay with **immediately killing** the container.

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
59e22667830b: Already exists
140da4f89dcb: Pull complete
96e47e70491e: Pull complete
2ef442a3816e: Pull complete
4b1e45a9989f: Pull complete
1d9f51194194: Pull complete
f30ffbee4c54: Pull complete
Digest: sha256:84ec966e61a8c7846f509da7eb081c55c1d56817448728924a87ab32f12a72fb
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
qwerty        latest    18935e99e185   3 hours ago    132MB
nginx         latest    2cd1d97f893f   2 weeks ago    192MB
python        latest    4bbf6d52fe8d   6 weeks ago    1.02GB
hello-world   latest    74cc54e27dc4   6 months ago   10.1kB
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker run --name ngx -d -p 8080:80 nginx
e589b32fcfe9fcbfe692d3116e1edddd09f7a2b196546ef0e7e3848c2d484d71
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                     NAMES
e589b32fcfe9   nginx     "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   ngx
9715289d1d92   python    "python3"                29 minutes ago   Up 29 minutes                                             sundari
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker rm 9715289d1d92
Error response from daemon: cannot remove container "9715289d1d92": container is running: stop the container before removing or force remove
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker stop 9715289d1d92
9715289d1d92
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker rm 9715289d1d92
9715289d1d92
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                     NAMES
e589b32fcfe9   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   ngx
```

docker rm -f [container-id]

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker rm -f e589b32fcfe9
e589b32fcfe9
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

12) Start or stop a running image -> docker stop/restart [image-name]

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    2cd1d97f893f   2 weeks ago    192MB
python        latest    4bbf6d52fe8d   6 weeks ago    1.02GB
hello-world   latest    74cc54e27dc4   6 months ago   10.1kB
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps -a
CONTAINER ID   IMAGE          COMMAND          CREATED             STATUS                          PORTS     NAMES
dcf39a1416d7   python         "python3"        8 minutes ago       Exited (0) 8 minutes ago                  modest_cerf
42626c622491   python         "python3"        About an hour ago   Exited (0) About an hour ago              admiring_bell
b9473ec7843e   18935e99e185   "python app.py"  About an hour ago   Exited (0) About an hour ago              newtester
998961e9b412   18935e99e185   "python app.py"  About an hour ago   Exited (0) About an hour ago              tester
07467a270267   hello-world    "/hello"         2 hours ago         Exited (0) 2 hours ago                    awesome_bhaskara
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker run --name sundari -it -d pyhton
Unable to find image 'pyhton:latest' locally
docker: Error response from daemon: pull access denied for pyhton, repository does not exist or may require 'docker login': denied: requested access to the resource is denied

Run 'docker run --help' for more information
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker run --name sundari -it -d python
88434aa5096d0574ffe5cbeaba3ac5ba13ba63ff514cb0cdc60841f116d477b0
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
88434aa5096d   python    "python3"   10 seconds ago   Up 10 seconds             sundari
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker stop sundari
sundari
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker restart sundari
sundari
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo> docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED             STATUS         PORTS     NAMES
88434aa5096d   python    "python3"   About a minute ago   Up 3 seconds             sundari
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo\demo>
```

13) remove/destor image or container -> docker rmi Image_Name and docker rm Container and one important point to note before deleting the image it respective container have to to removed first

14) Port binding -> docker run -p <host_port>:<container_port> image_name

📦 Example: MySQL Container

bash Copy Edit

```
docker run -d -p 3306:3306 --name mysql-container mysql:8.0
```

Part	Meaning
`-p 3306:3306`	Binds **host port 3306** to **container port 3306**
`mysql:8.0`	MySQL is listening on port **3306** internally

🔁 You Can Map Different Ports

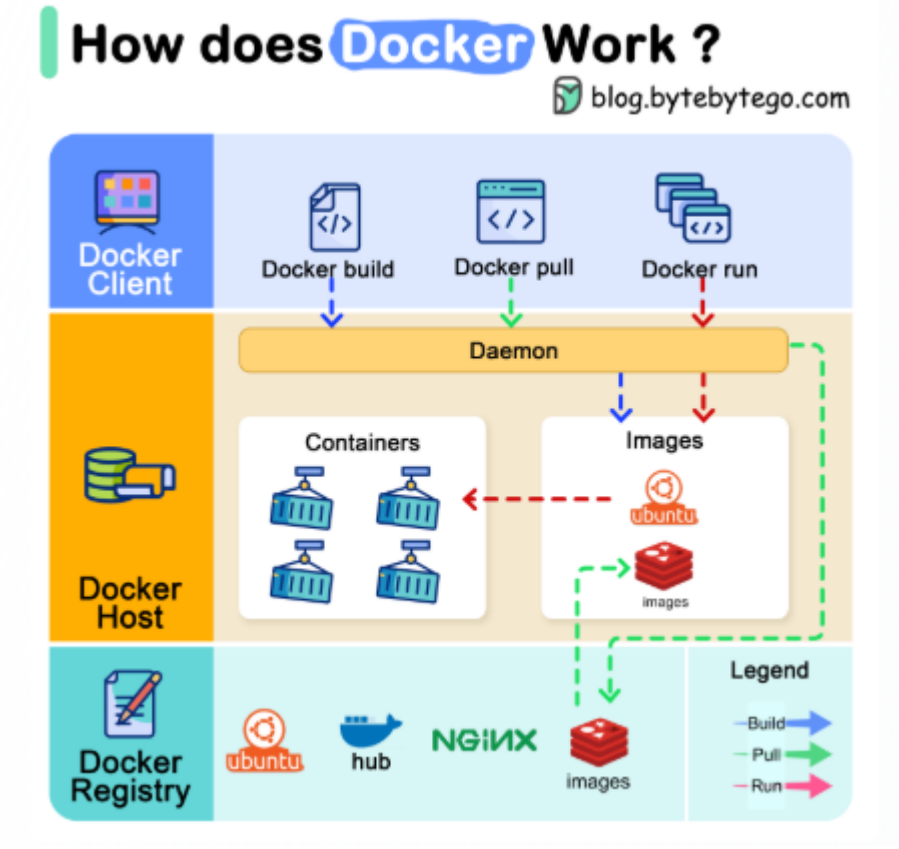
bash Copy Edit

```
docker run -p 8080:80 nginx
```

Host Port	Container Port	Description
8080	80	Host port 8080 sends traffic to container's port 80 (e.g., Nginx web server)

Access via: `http://localhost:8080`

How Docker works ?



The diagram below shows the architecture of Docker and how it works when we run “docker build”, “docker pull” and “docker run”.

There are 3 components in Docker architecture:

Docker client

The docker client talks to the Docker daemon.

Docker host

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

Docker registry

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use.

`Let’s take the “docker run” command as an example.`

Docker pulls the image from the registry. Docker creates a new container. Docker allocates a read-write filesystem to the container. Docker creates a network interface to connect the container to the default network. Docker starts the container.

Difference between VM and Docker

The key difference lies in the level of virtualization.

- **Docker (Containers):**
  - Virtualizes at the **application layer**.
  - Uses the **host OS kernel** directly.
  - Each container shares the host OS kernel but has its own user space (libraries, dependencies, etc.).
  - **Lightweight** and faster to start.
- **Virtual Machines (VMs):**
  - Virtualize the **entire hardware stack**, including the **kernel**.
  - Each VM runs a **full guest OS**, including its own kernel.
  - **Heavier** and slower to boot compared to containers.

Docker Compose

It is a tool for defining and running multi-container applications. When your app needs multiple containers (like a web server + database + cache). Docker Compose helps manage them all together.

For example: `A Flask app / A PostgreSQL database / A Redis cache. You can define all three in one docker-compose.yml`

Here's a simple docker-compose.yml example for a web app using Node.js and MongoDB, followed by a line-by-line explanation.

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - mongo
    environment:
      - MONGO_URL=mongodb://mongo:27017/mydb

  mongo:
    image: mongo:6
    volumes:
      - mongo-data:/data/db

volumes:
  mongo-data:
```

🧠 Explanation

◇ version: '3.8'

Specifies the version of the Docker Compose file format. Version 3.8 is commonly used and supports modern features.

- services:

Defines the containers your app needs. In this case: app and mongo.

1. app Service (Your Node.js App)

```
app:
    build:
```

Tells Docker to build an image for the app using the Dockerfile in the current directory.

```
ports:
    - "3000:3000"
```

Maps host port 3000 to container port 3000, so you can access the app at localhost:3000.

```
depends_on:
  - mongo
```

Ensures the mongo container starts before the app container.

```
environment:
  - MONGO_URL=mongodb://mongo:27017/mydb
```

Sets an environment variable inside the container, typically used by the Node app to connect to MongoDB.

2. mongo Service (MongoDB Database)

```
mongo:
    image: mongo:6
```

Uses the official MongoDB image (version 6) from Docker Hub.

```
volumes:
  - mongo-data:/data/db
```

Mounts a named volume (mongo-data) to persist MongoDB data even if the container stops or is deleted.

3. volumes:

```
    volumes:
        mongo-data:
```

Declares a named volume to be used by the mongo service. This ensures database data is stored outside the container's filesystem.

Summary

This setup runs:

A Node.js app that connects to...

A MongoDB database, using a volume to persist data.

We can run it all with:

```
docker-compose up
```

Docker Compose Command

Basic Commands

Command	Description
`docker-compose up`	**Builds**, (if needed) and **starts** all containers.
`docker-compose up -d`	Same as above, but runs containers in **detached** (background) mode.
`docker-compose down`	**Stops and removes** all containers, networks, and volumes defined in the file.

Start, Stop, Restart

Command	Description
`docker-compose start`	Starts **existing** stopped containers.
`docker-compose stop`	Stops running containers **without removing** them.
`docker-compose restart`	Restarts all containers.

🔍 Monitor Containers

Command	Description
`docker-compose ps`	Lists containers and their status.
`docker-compose logs`	Shows logs for all services.
`docker-compose logs -f`	Follows logs (like `tail -f`).
`docker-compose top`	Shows running processes inside containers.

🧪 Run Commands in Containers

Command	Description
`docker-compose exec <service> <command>`	Runs a command inside a **running** container (like `bash`, `npm install`).
`docker-compose run <service> <command>`	Starts a **new container** and runs the command (not tied to current one).

🧹 Clean Up

Command	Description
`docker-compose down -v`	Removes containers **and volumes**.
`docker-compose rm`	Removes stopped service containers.
`docker-compose down --rmi all`	Removes containers **and images**.

🧰 Advanced

Command	Description
`docker-compose config`	Validates and prints the final **combined config**.
`docker-compose pull`	Pulls service images from Docker Hub.
`docker-compose push`	Pushes built service images to a registry.
`docker-compose scale <service>=<num>`	Scales service to a specific number of containers (only in v2 syntax).

🌐 What is Docker Network?

Docker networking allows containers to communicate with

1. Each other without needing any kind of port or local host
2. The host system

3. The outside world (internet)

Docker Network Drivers

When you run:

```bash
docker network ls
```

You'll see default networks like:

NAME	DRIVER
bridge	bridge
host	host
none	null

1 bridge (Default for containers)

- **Default network** for standalone containers.
- Creates a **private internal network** on your host.
- Docker **assigns IPs** to containers.
- Containers can **talk to each other by name** (if on same network).
- Useful for **most apps**, like databases, APIs, frontends.

Example:

```bash
docker network create my-custom-network
```

→ This creates a **bridge network by default**.

2 host

- Container **shares the host's network stack**.
- No **network isolation** — uses host's IP and ports directly.
- No need to expose ports using `-p`, because the container is already on host network.

Example:

```bash
docker run --network host nginx
```

Limitations:

- **Only works on Linux.** (Not supported on Docker Desktop for Mac/Windows)
- No container-to-container isolation.

3 null (none)

- **Disables networking** entirely.
- No inbound/outbound traffic.
- Useful for testing or extreme isolation.

Example:

```bash
docker run --network none alpine
```

🤔 Why Does a Custom Network Use `bridge` by Default?

When you run:

```bash
docker network create my-net
```

- Docker assumes you want a **typical isolated network**.
- `bridge` driver provides **container-to-container communication** + isolation from the host.
- It's **flexible and secure** — that's why it's default.

To specify a different driver:

```bash
docker network create --driver host my-host-net
```

Docker network command

1) Lists all the Docker networks available on your system. docker network ls

2) Creates a new user-defined bridge network. docker network create Network_Name

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
7e2f75516dff   bridge    bridge    local
3325d3c48dfa   host      host      local
c33c60a9aece   none      null      local
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker network create mongo-network
c408b5abfa4efc97aee17cb9e4d774e599d8129c9d7f2fc2f010117bb070eb3c
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
7e2f75516dff   bridge          bridge    local
3325d3c48dfa   host            host      local
c408b5abfa4e   mongo-network   bridge    local
c33c60a9aece   none            null      local
```

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker run -d `
>>   -p 27017:27017 `
>>   --name mongo `
>>   --network mongo-network `
>>   -e MONGO_INITDB_ROOT_USERNAME=admin `
>>   -e MONGO_INITDB_ROOT_PASSWORD=qwerty `
>>   mongo
>>
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
32f112e3802c: Already exists
93ce1ce52c19: Pull complete
042b69506939: Pull complete
75d3308c8da5: Pull complete
be1ecb1b5145: Pull complete
67bb4a146246: Pull complete
e6d0bf56c1cc: Downloading [=======>                                           ]  40.78MB/263.6MB
4e58ed1e1297: Download complete
```

✅ Result

This command runs a MongoDB container:

- In the background
- On port 27017
- With admin credentials
- Connected to the custom Docker network `mongo-network`
- Named `mongo` for easy reference

Explanation: ▶ `docker run -d` docker run: Runs a new container.

-d: Detached mode – runs the container in the background.

▶ `-p 27017:27017` Maps port 27017 on your host to port 27017 inside the container.

MongoDB uses port 27017 by default, so this makes it accessible from your machine.

▶ `--name mongo` Names the container mongo so you can easily reference it (instead of using the container ID).

▶ `--network mongo-network` Connects the container to a user-defined Docker network called mongo-network.

This allows containers in the same network to communicate by name (like mongo, mongo-express, etc.).

⚠ Make sure this network exists. If not, create it with:

docker network create mongo-network ▶ `-e MONGO_INITDB_ROOT_USERNAME=admin` Sets the root username for MongoDB to admin.

▶ `-e MONGO_INITDB_ROOT_PASSWORD=qwerty` Sets the root password for MongoDB to qwerty.

These credentials will be used to log into MongoDB securely.

▶ `mongo` Specifies the Docker image to use – in this case, the official MongoDB image from Docker Hub.

Mongo express

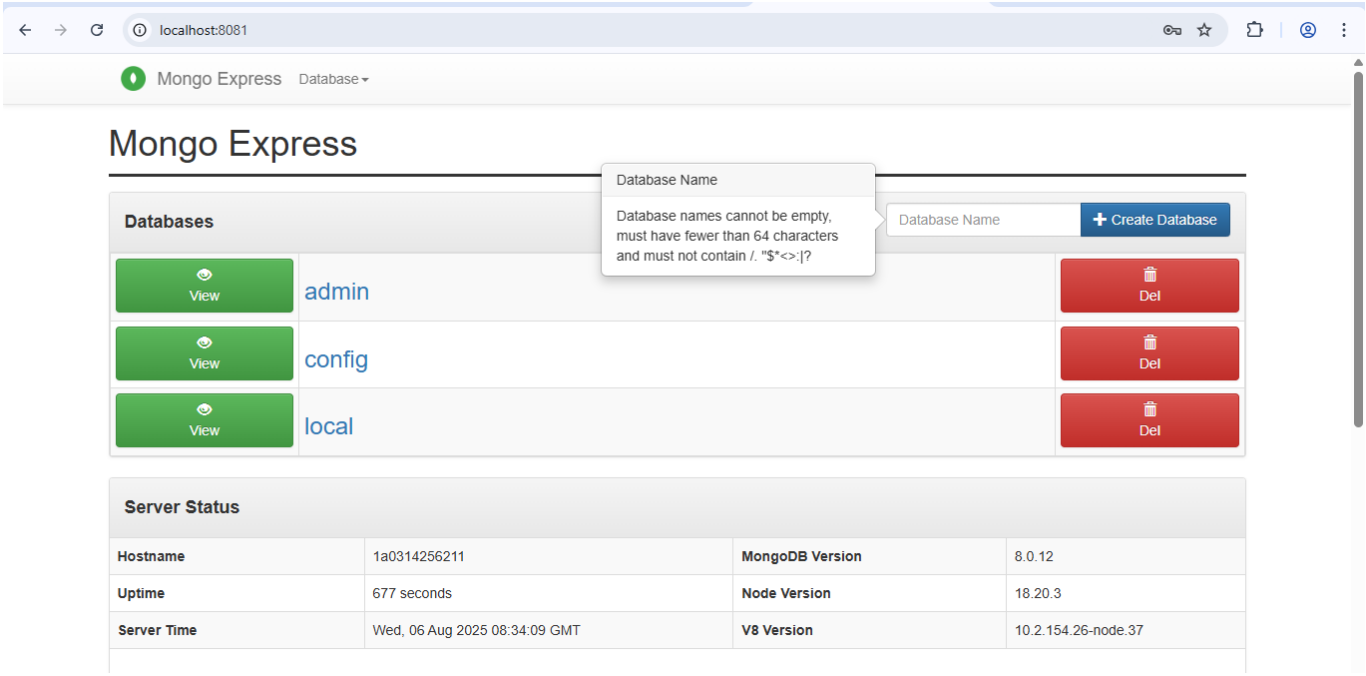
It is the GUI for mongo db provided by docker hub.

```powershell
docker run -d `
  -p 8081:8081 `
  --name mongo-express `
  --network mongo-network `
  -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin `
  -e ME_CONFIG_MONGODB_ADMINPASSWORD=qwerty `
  -e ME_CONFIG_MONGODB_URL="mongodb://admin:qwerty@mongo:27017" `
  mongo-express
```

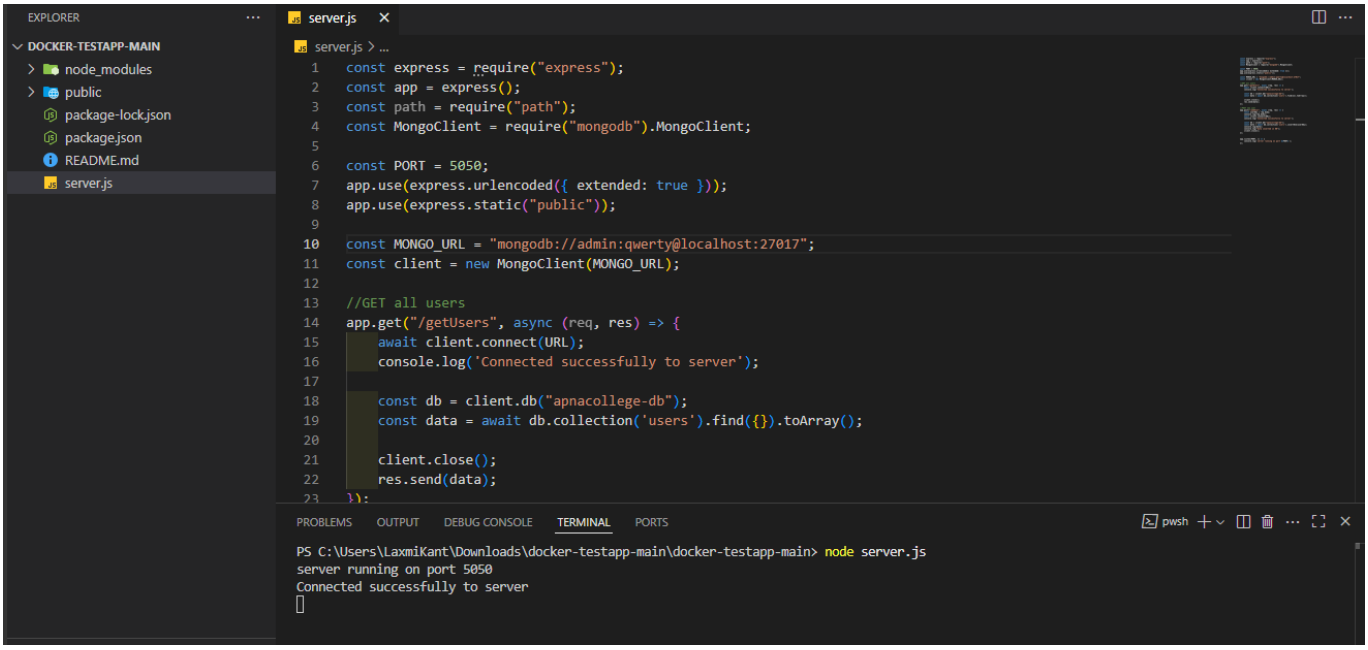
🧠 **Explanation:**

Part	Purpose
`docker run -d`	Runs the container in detached mode.
`-p 8081:8081`	Maps port 8081 on your host to 8081 in the container (Mongo Express UI).
`--name mongo-express`	Names this container for easy reference.
`--network mongo-network`	Connects to the same network as MongoDB (`mongo` container).
`-e ME_CONFIG_MONGODB_ADMINUSERNAME=admin`	MongoDB admin username.
`-e ME_CONFIG_MONGODB_ADMINPASSWORD=qwerty`	MongoDB admin password.
`-e ME_CONFIG_MONGODB_URL="mongodb://admin:qwerty@mongo:27017"`	Connection string to the MongoDB container named `mongo`.
`mongo-express`	Uses the official Mongo Express Docker image.

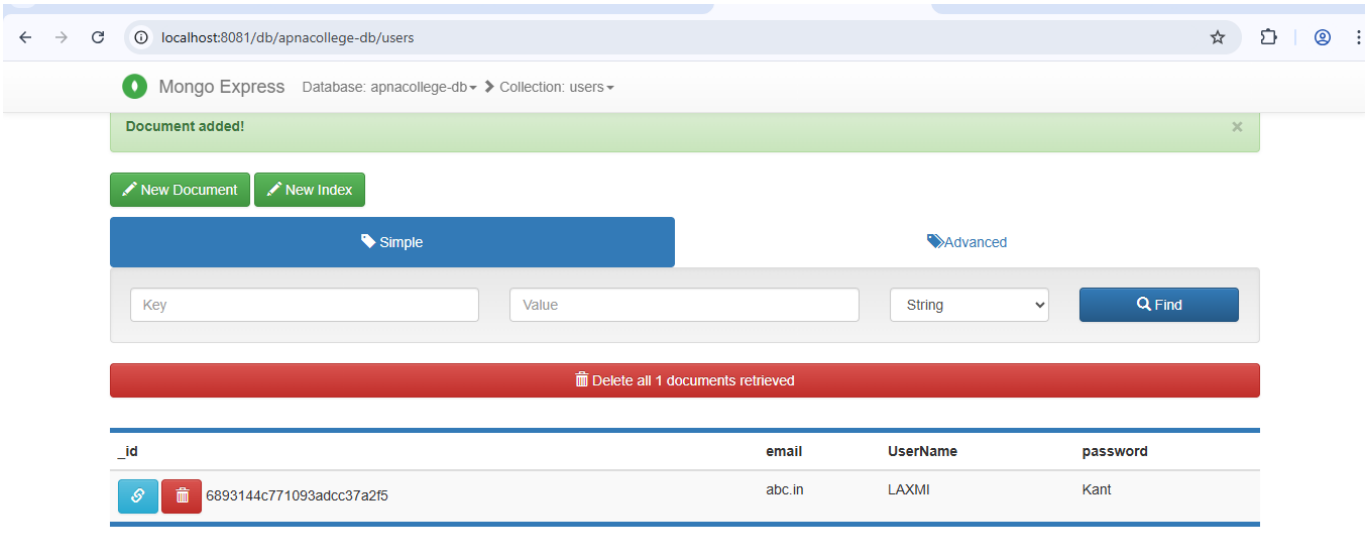
we can acces the Mongo db gui by accesing localhost:8081



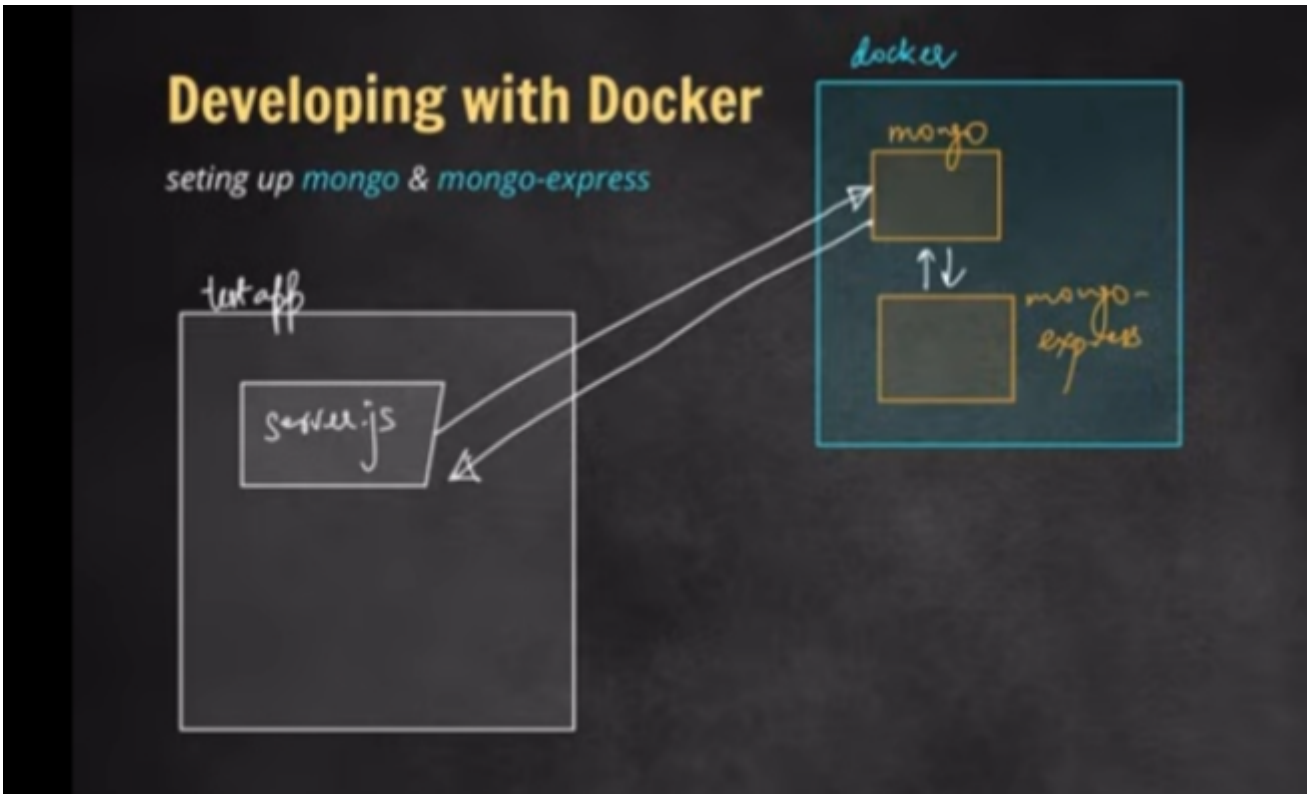
Below is the node js application is running on the port number 5050



after creating our own database name as apna college-db and having collections named as users



Onece the network connection is established we can able to acces the data from the mongodb which is running on port number 57017



[{"_id":"6893144c771093adcc37a2f5","email":"abc.in","UserName":"LAXMI","password":"Kant"}]

DOCKER COMPOSE

Docker Compose lets you define and run multi-container applications using a simple docker-compose.yml file.

Great observation — and this is an **excellent question**! You're absolutely right:

- In the `docker run` command, you **manually define the network** using `--network mongo-network`.
- In the `docker-compose.yml` file, you **don't need to create the network manually** — Docker Compose does it for you.

🤔 So, why the difference?

✅ 1. When using `docker run` (CLI)

You must **manually create the network first**, like this:

```bash
docker network create mongo-network
```

Then, when running your container, you attach it using:

```bash
--network mongo-network
```

> 🧠 Docker CLI doesn't manage networks automatically — it's up to **you** to create and connect containers to a network.

✅ 2. When using Docker Compose

Docker Compose **automatically creates and manages the network** for you, unless you explicitly tell it not to.

In your `docker-compose.yml` file:

```yaml
networks:
  app-network:
    driver: bridge
```

If this block is present, Compose:

- Creates the network (`app-network`)
- Connects all services to it automatically
- You don't need to run `docker network create` manually

Even if you **don't** include the `networks:` section at all, Docker Compose will still create a **default network** (named like `<project-name>_default`) and connect all services to it.

There are two ways Docker Compose handles networks:

🔹 1. Explicit network definition (what your file does)

You define the network like this:

```yaml
networks:
  app-network:
    driver: bridge
```

And attach services to it:

```yaml
services:
  app:
    networks:
      - app-network
```

✅ This gives you:

- Custom **network name**
- Option to configure the driver (like `bridge`, `overlay`, etc.)
- **Reusable** and **clear** network config

🔹 2. Implicit (default) network

If you remove the whole `networks:` section, Docker Compose still works. It just creates a **default network** named:

```objectivec
<project-name>_default
```

And all services are automatically attached to it **unless you specify otherwise**.

Example:

```yaml
services:
  mongo:
    image: mongo
```

🟢 This works, but you can't control the network name or behavior easily.

✔ You did define a custom network (`app-network`) in the YAML.

This is better than using the default one because:

- It gives **predictable naming**
- Makes the network **reusable**
- You can extend it (e.g., connect external services)

🆚 Compared to `docker run`

In `docker run`, you **must**:

1. Manually create the network (`docker network create mongo-network`)
2. Manually attach it to each container (`--network mongo-network`)

In **Docker Compose**, you:

- Define the network **once** in YAML
- Docker handles the creation and attachment **automatically**

REASON

📦 Example: What Happens with `docker run`

If you use:

powershell Copy Edit

```
docker run -d `
  -p 8081:8081 `
  --name mongo-express `
  --network mongo-network `
  -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin `
  -e ME_CONFIG_MONGODB_ADMINPASSWORD=qwerty `
  -e ME_CONFIG_MONGODB_URL="mongodb://admin:qwerty@mongo:27017" `
  mongo-express
```

You are:

- Typing a long, error-prone command.
- Relying on manual order (Mongo container must start before this).
- Manually managing the network.
- Repeating it every time you restart or share it.

✅ Why Docker Compose Is Better

With Docker Compose, you do **everything above** with **one command** and a **YAML file**:

bash Copy Edit

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

The YAML file holds:

- The image
- Port mappings
- Environment variables
- Network config
- Dependencies (`depends_on`)

You can share this file with your team or deploy it on another machine **without retyping any commands**.

DIFFERENCE ?

Docker CLI (`docker run`) vs. Docker Compose

Feature	`docker run` (CLI)	Docker Compose
Ease of use	Manual, needs long commands	Declarative config file
Multi-container support	Needs multiple commands	Designed for multi-container apps
Reusability	Hard to repeat or share	One file (`docker-compose.yml`) can be version-controlled
Environment variables	Passed manually each time	Can be stored in `.env` or in YAML
Networking	Must create and manage networks manually	Networks are created automatically
Scaling containers	Complex to scale	One command: `docker-compose up --scale app=3`
Build + Run	Manual build and run	Automatically builds and runs services
Volume/config management	Manual	Integrated and declarative
Logs & lifecycle	Separate commands per container	`docker-compose logs`, `stop`, `restart` handles all

WHAT TO USE

Summary

CLI `docker run` is good for:	Docker Compose is better for:
Quick, one-off containers	Multi-container apps (Node.js + MongoDB)
Simple tasks or testing	Reproducible, sharable setups
Scripting	Long-term development workflows
Beginners learning Docker	Teams and production setups

example : -

docker-compose.yml

```
version: '3.8'  # Compose file version

services:
  app:  # Node.js application
    build: .  # Build image from Dockerfile in current directory
    container_name: node-app
    ports:
      - "3000:3000"  # Host:Container port mapping
```

```
    environment:
      - MONGO_URL=mongodb://admin:qwerty@mongo:27017  # Used by app to connect to
MongoDB
    depends_on:
      - mongo  # Ensure mongo starts before app
    networks:
      - app-network

  mongo:  # MongoDB service
    image: mongo  # Official MongoDB image
    container_name: mongo
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=qwerty
    networks:
      - app-network

  mongo-express:  # Web-based MongoDB UI
    image: mongo-express
    container_name: mongo-express
    ports:
      - "8081:8081"
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
      - ME_CONFIG_MONGODB_ADMINPASSWORD=qwerty
      - ME_CONFIG_MONGODB_URL=mongodb://admin:qwerty@mongo:27017
    depends_on:
      - mongo
    networks:
      - app-network

# Network definition for inter-container communication
networks:
  app-network:
    driver: bridge
```

📁 Folder structure should look like:

```
go

your-project/
├── app.js
├── package.json
├── Dockerfile
└── docker-compose.yml
```

🚀 How to Run It:

From your project folder (where this YAML file is saved):

1. **Build and start everything:**

```
bash

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

2. **Access your apps:**
   - Node.js: http://localhost:3000
   - Mongo Express UI: http://localhost:8081
3. **Stop everything:**

```
bash

docker-compose down
```

🔁 What It Demonstrates:

- **Multi-container setup** with Node.js, MongoDB, and Mongo Express
- Automatic **networking** between services using `app-network`
- Easy **environment variable management**
- `depends_on` ensures Mongo starts before the ot services
- Portable, sharable configuration for teams

Docker compose command

```
docker compose -f fileName.yml up -d
docker compose -f fileName.yml down
```

▶️ `docker compose -f fileName.yml up -d`

Part	Meaning
`docker compose`	Tells Docker to use **Compose** (v2 syntax).
`-f fileName.yml`	Specifies the **Compose file** to use (e.g., `docker-compose.prod.yml`)
`up`	**Starts** and creates containers, builds images if needed
`-d`	Runs in **detached mode** (in the background)

📦 **Example:**

bash Copy Edit

```
docker compose -f docker-compose.prod.yml up -d
```

➡️ Starts all services defined in `docker-compose.prod.yml` in the background.

⏹️ `docker compose -f fileName.yml down`

Part	Meaning
`docker compose`	Again, using Docker Compose
`-f fileName.yml`	Refers to the same file
`down`	**Stops and removes** all containers, networks, and volumes created by `up`

📦 **Example:**

bash Copy Edit

```
docker compose -f docker-compose.prod.yml down
```

➡️ Stops all services and cleans up the containers and network created by that file.

NOTE : Now we are going to run the two container in the same network using the yml and docker compose.

1. Created qwerty.yaml

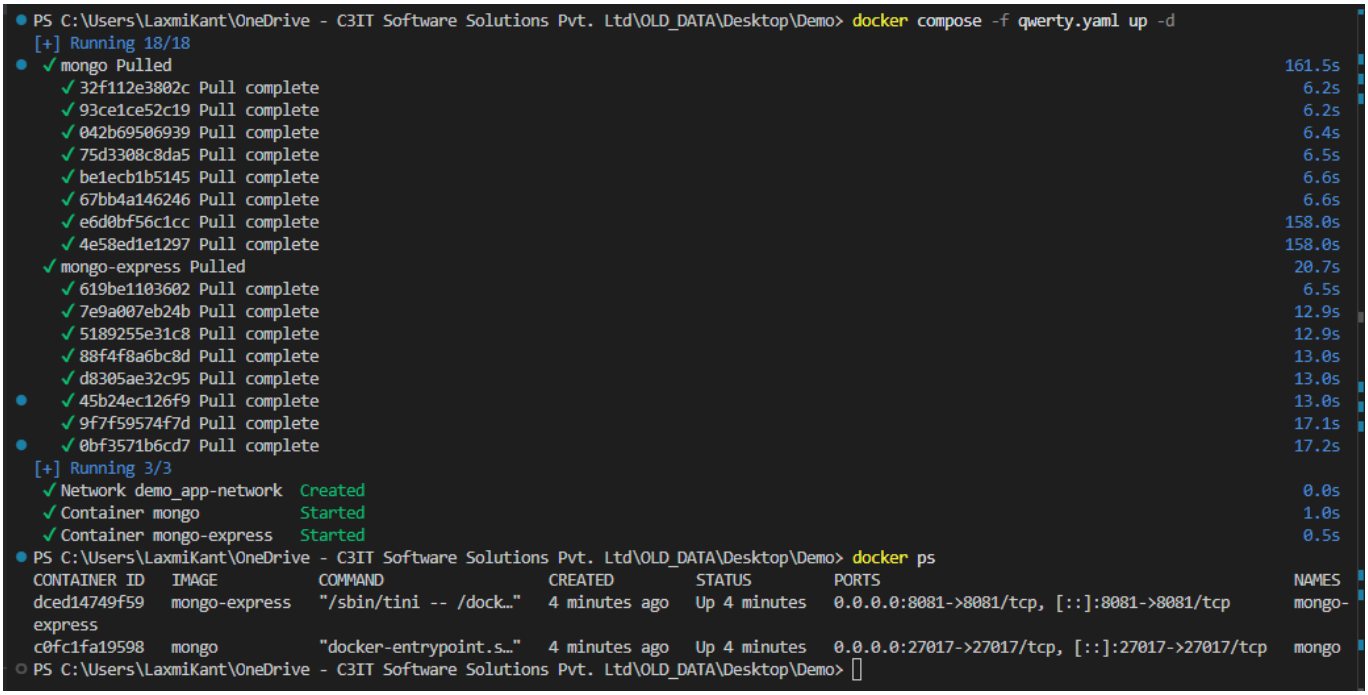
```
services:
  mongo:  # MongoDB service
    image: mongo
    container_name: mongo
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=qwerty
    networks:
      - app-network

  mongo-express:  # Web-based MongoDB UI
```

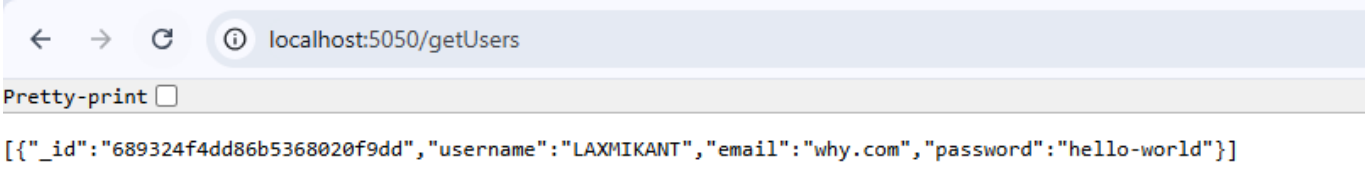
```
    image: mongo-express
    container_name: mongo-express
    ports:
      - "8081:8081"
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
      - ME_CONFIG_MONGODB_ADMINPASSWORD=qwerty
      - ME_CONFIG_MONGODB_URL=mongodb://admin:qwerty@mongo:27017
    depends_on:
      - mongo
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

2. Running the container



We are able to connect the node application with our container mongodb database using docker compose



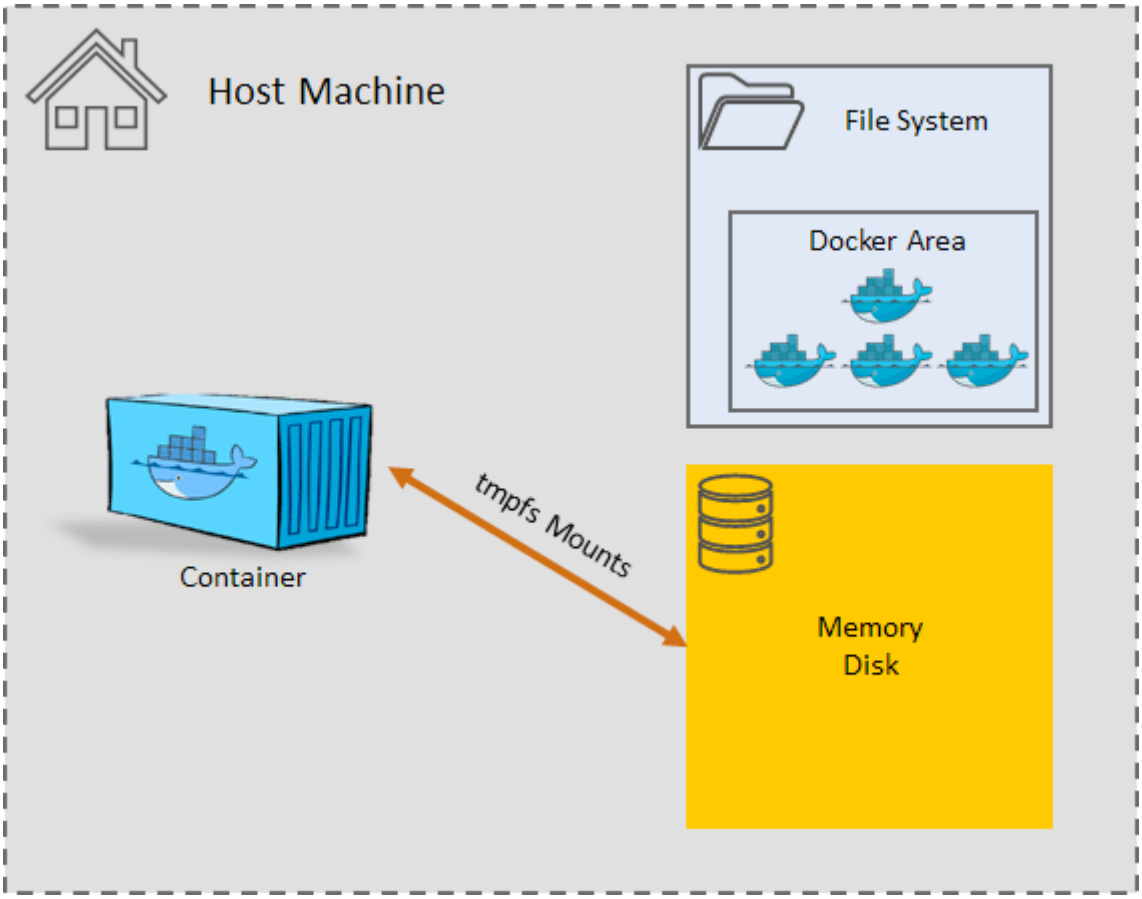
3. Removing the running network , applictaion in the container

```
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker compose -f qwerty.yaml down
[+] Running 3/3
 ✓ Container mongo-express  Removed
 ✓ Container mongo          Removed
 ✓ Network demo_default     Removed
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\LaxmiKant\OneDrive - C3IT Software Solutions Pvt. Ltd\OLD_DATA\Desktop\Demo> docker network
Usage:  docker network COMMAND

Manage networks
```

Docker volume

Volumes are persistent data store for containers



A volume in Docker is a special directory stored outside the container's filesystem, used to:

1. ☑ Persist data even when containers are deleted.
2. 🔁 Share data between multiple containers.
3. 🗁 Keep the container lightweight and stateless.

🧱 Using Volumes in a Container

✅ Basic Volume Mount (Named Volume)

bash Copy Edit

```
docker run -d \
  --name mongo \
  -v mongo-data:/data/db \
  mongo
```

- `mongo-data` is the **volume name**.
- `/data/db` is the **path inside the container** where MongoDB stores data.
- This ensures **MongoDB data persists** even if the container is deleted.

✅ Bind Mount (Mount Local Folder)

bash Copy Edit

```
docker run -it \
  -v /your/local/path:/app/data \
  ubuntu
```

- Maps a **host folder** (`/your/local/path`) to `/app/data` inside the container.
- You can access local files inside the container and vice versa.

commands

1. NAMED VOLUMES

`docker run -v VOL_NAME:CONT_DIR`

📦 1) Named Volumes

✅ Syntax:

bash Copy Edit

```
docker run -v volume_name:/container/path
```

✅ Example:

bash Copy Edit

```
docker run -d -v mydata:/data busybox
```

✅ What it does:

- Docker **creates (or reuses) a volume named** `mydata` .
- Mounts it inside the container at `/data` .
- **Data persists** across container deletions and can be **reused** by other containers.

✅ Use Case:

> Good for **persistent data**, like databases (e.g., MongoDB, MySQL).

2. ANONYMOUS VOLUMES

`docker run -v MOUNT_PATH`

2) Anonymous Volumes

Syntax:

```bash
docker run -v /container/path
```

Example:

```bash
docker run -d -v /data busybox
```

What it does:

- Docker **creates a random volume name** (not user-defined).
- Mounts it to `/data` inside the container.
- Data **persists**, but you can't easily reference the volume by name later.

Use Case:

> Useful when **you don't care** about reusing or nam the volume.

3. BIND MOUNT

`docker run -v HOST_DIR:CONT_DIR`

3) Bind Mounts

Syntax:

```bash
docker run -v /host/path:/container/path
```

Example:

```bash
docker run -d -v /home/user/logs:/var/log nginx
```

What it does:

- Mounts a **host machine directory** (`/home/user/logs`) into the container.
- Any changes made inside the container are reflected on the host (and vice versa).
- Gives **full access** to host files.

Use Case:

> Great for **development**, **logging**, or using **host config files**.

4. DELETE UNUSED VOLUMES

```
docker volume prune
```

difference

🧱 Types of Docker Volumes

Feature	Named Volume	Anonymous Volume	Bind Mount
👄 Name	You define a name (e.g., `my-vol`)	Docker assigns a random name	No name — uses host directory
📁 Location	Stored in Docker-managed folder (`/var/lib/docker/volumes/...`)	Same as named	Any specific path on the host
🔁 Reusable	✅ Yes (by name)	❌ No (hard to reuse)	✅ Yes (by directory path)
📦 Persistence	✅ Yes	✅ Yes	✅ Yes
🔗 Host File Access	❌ No direct access	❌ No direct access	✅ Full access
🔐 Security	🔒 Safe and isolated	🔒 Isolated, but unmanaged	⚠️ Less secure (host exposure)
🛠️ Best for	Databases, reusable storage	Temporary data, quick tests	Development, config files, logs