1️⃣ What is Docker Compose

- **Docker** = tool to package your app & environment into a "container."
- **Docker Compose** = a tool to **run multiple containers together** (multi-container apps) using a single YAML file (`docker-compose.yml`).

Think of Compose as a **"movie director"** telling Docker "start the database first, then the backend, link them, give them networks."

2️⃣ Why Docker Compose is Useful

Problem (without Compose)	Solution (with Compose)
You run many `docker run ...` commands manually	One `docker-compose up` launches all containers
Hard to manage networks, volumes manually	Compose auto-creates networks & volumes from YAML
Different environments (dev/prod) confusing	One YAML can define services & environment variables clearly

---

3️⃣ Core Concepts

Term	Meaning (easy)
Service	A container definition (like "app" or "db").
Image	The Docker image to use for the service.
Container	A running instance of a service.
Volume	Persistent storage for a service.
Network	A virtual network so services can talk to each other by name.
Environment Vars	Settings passed into the container (like DB password).

4️⃣ Commands Cheat Sheet

```
docker-compose up        # start all services (in foreground)
docker-compose up -d      # start in background
docker-compose down       # stop & remove all containers
docker-compose ps        # see running containers
docker-compose logs       # view logs of all services
docker-compose build      # build images defined in the YAML
```

5️⃣ Project: Node.js + MongoDB with Docker Compose

This will give you:

- **Backend**: Node.js Express app
- **Database**: MongoDB container

You'll be able to run `docker-compose up` and see your app connect to MongoDB automatically.

5.1 Folder Structure

```
docker-compose-demo/
  ├── app/
  │   ├── Dockerfile
  │   ├── package.json
  │   └── server.js
  └── docker-compose.yml
```

5.2 Create `app/package.json`

```
{
  "name": "docker-compose-demo",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "express": "^4.18.2",
```

```
    "mongoose": "^7.6.0"
  }
}
```

Run `npm install` inside `app/` if you're testing locally (optional).

5.3 Create `app/server.js`

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

const app = express();
app.use(express.json());

// Connect to MongoDB (the hostname 'mongo' matches service name in com
pose)
mongoose.connect('mongodb://mongo:27017/mydb')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error(err));

app.get('/', (req, res) => {
  res.send('Hello from Node + Mongo running in Docker Compose!');
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

5.4 Create `app/Dockerfile`

```
# Use Node official image
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
```

```
RUN npm install

COPY . .

EXPOSE 3000
CMD ["node", "server.js"]
```

5.5 Create `docker-compose.yml` (in project root)

```
version: '3.8'

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

  mongo:
    image: mongo:6.0
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db

volumes:
  mongo_data:
```

5.6 Steps to Run

1. Save all files.
2. In your project root, run:

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

3. Wait until logs show:
   - “Connected to MongoDB” (from Node)
   - MongoDB container started
4. Open browser at `http://localhost:3000` → should see “Hello from Node + Mongo running in Docker Compose!”

5.7 Verify Everything Works

- `docker-compose ps` → shows `app` and `mongo` running
- `docker exec -it <mongo_container> mongosh` → you can connect to MongoDB inside container
- Modify `server.js` and refresh page, changes reflected instantly (because of volume mount)

6️⃣ Recap: What You Learned

- **Docker Compose** groups multiple containers & runs them with one command.
- You defined **services**, **volumes**, **ports**, **dependencies** in YAML.
- You built a real **Node + Mongo project** running fully in containers.
