2-1 What are Images & Containers?

🔹 **Image** = Blueprint
A Docker image is like a **recipe or template**. It includes the app code + dependencies + environment. You don't run an image directly.

🔹 **Container** = Running Instance
A Docker container is a **running version** of the image. You can start, stop, restart it — it's like launching a program from a template.

🧠 **Analogy:**
Think of an image like a class in programming and a container like an object (instance of that class).

2-2 Using Pre-built Images

Docker has **official ready-made images** for common tools — Node, Python, MySQL, etc.

📦 **Example: Running a Node.js REPL**

bash
CopyEdit
```
docker run -it node
```

🧠 **Explanation:**

- `docker run`: Start a container
- `-it`: Interactive mode (so we can type inside)
- `node`: Use the pre-built Node image from Docker Hub

You're now inside a Node REPL running inside a container!

2-3 Writing Our First Dockerfile

Let's say you have a basic Node.js app:

📁 **Folder structure:**

pgsql
CopyEdit

```
app/
  ├── index.js
  └── package.json
```

📄 **index.js**

js
CopyEdit

```
console.log("Hello from Docker!");
```

📄 **Dockerfile**

Dockerfile
CopyEdit

```
# 1. Use base image
FROM node:18

# 2. Set working directory inside container
WORKDIR /app

# 3. Copy files to container
COPY . .

# 4. Install dependencies
RUN npm install

# 5. Define default command
CMD ["node", "index.js"]
```

🧠 **Explanation:**

- `FROM node:18`: Use Node.js v18 image as the base
- `WORKDIR /app`: Inside the container, we'll work in `/app`

- `COPY . .`: Copy everything from current folder into container
- `RUN npm install`: Install dependencies
- `CMD`: The command to run when container starts

2-4 Building an Image & Running Container Based On Our Image

🛠️ Build the image:

bash
CopyEdit
```
docker build -t my-node-app .
```

🏃 Run the container:

bash
CopyEdit
```
docker run my-node-app
```

🧠 **Explanation:**

- `-t my-node-app`: Tag the image with a name
- `.`: Use the current directory as context (Dockerfile is here)
- The `run` command creates a container from the image and runs the default command (`node index.js`)

Output should be:

csharp
CopyEdit
```
Hello from Docker!
```

2-5 Deep Dive Into Docker Images

- Images are **built in layers**. Each command in Dockerfile (`FROM`, `COPY`, etc.) creates a new layer.
- Layers are **cached** — speeds up rebuilds.
- Images are **immutable** — once built, they don't change.

🔍 See image details:

bash
CopyEdit
```
docker image inspect my-node-app
```

🧠 **Good to Know:**

- Docker layers = fast, smart builds.
- Small changes don't rebuild everything.

2-6 Managing Images & Containers

🧹 See images:

bash
CopyEdit
```
docker images
```

🗑️ Remove image:

bash
CopyEdit
```
docker rmi my-node-app
```

📦 See containers (running only):

bash
CopyEdit
```
docker ps
```

📦 See all containers (including stopped):

bash
CopyEdit
```
docker ps -a
```

🗑️ Remove a container:

bash
CopyEdit
```
docker rm container_id
```

2-7 Attached & Detached Container

Attached mode (default) = You see container logs/output.

bash
CopyEdit
```
docker run my-node-app
```

Detached mode = Runs in background.

bash
CopyEdit
```
docker run -d my-node-app
```

🧠 **Use detached mode** when:

- You don't need to see logs live
- You want it to run like a background service

2-8 Deep Dive Into Container

Each container:

- Has its own **filesystem**, **IP address**, **processes**
- Is isolated but can communicate with other containers (via networks)

🛠️ Check inside a running container:

bash
CopyEdit
```
docker exec -it container_id sh
```

🧠 `exec` = Run command inside a running container
`-it` = Interactive shell
`sh` = Open shell (like a terminal inside the container)

2-9 Naming & Tagging Container & Images

🖊️ You can tag images like versions:

bash
CopyEdit
```
docker build -t my-node-app:v1 .
```

🧠 **Why tag?**

- Helps you manage versions
- Example: `my-node-app:v1`, `:v2`, `:latest`

📛 Run container with a name:

bash
CopyEdit
```
docker run --name my-container my-node-app
```

🧠 **Named containers** are easier to manage:

bash
CopyEdit

```
docker stop my-container
docker start my-container
```

2-10 Pushing Docker Image to DockerHub

🧰 Step-by-step:

1. Create account on https://hub.docker.com

Login:

bash
CopyEdit
```
docker login
```

2.

Tag image for Docker Hub:

bash
CopyEdit
```
docker tag my-node-app yourusername/my-node-app:v1
```

3.

Push image:

bash
CopyEdit
```
docker push yourusername/my-node-app:v1
```

4.

Now anyone can pull and use your image.

2-11 Pulling Docker Image From DockerHub

To download (pull) someone's image:

bash

CopyEdit

```
docker pull yourusername/my-node-app:v1
```

Run it:

bash
CopyEdit

```
docker run yourusername/my-node-app:v1
```

🧠 You can share your app like this with the world — just give them your Docker Hub repo name.