🐳 Docker: Complete Essential Guide for Java Developers

✅ What Is Docker?

Docker is a platform that lets you **package, run, and distribute your applications** in lightweight **containers**.

🔹 Why Do We Use Docker?

Purpose	Explanation
✅ **Environment Consistency**	Same app runs anywhere (dev, QA, prod)
✅ **Isolation**	Each app runs independently in a container
✅ **Faster Deployment**	Containers are fast to start & stop
✅ **Less Conflicts**	Avoids "It works on my machine" issues
✅ **CI/CD Friendly**	Easy to integrate with Jenkins, GitHub Actions

---

🚀 Real-Life Use Case:

You built a Spring Boot + MySQL app on your laptop.
With Docker, you can package **both app and database** into containers and run them on **any server**, exactly the same way.

🧱 Docker Core Concepts

---

1. 🧊 Image

- A **read-only template** with app code + dependencies + OS.
- Like a **blueprint** to create containers.

📝 Example:
openjdk:17 → Official Java image
mysql:8.0 → MySQL image

---

2. 📦 Container

- A **running instance of an image**.
- You run your app **inside a container**.

📝 Example:
docker run -d -p 8080:8080 myapp

---

3. 🏷️ Dockerfile

- A text file to build a custom Docker image.

📜 Sample Dockerfile for Spring Boot:

Dockerfile

CopyEdit

FROM openjdk:17

COPY target/student-api.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]

---

4. 🔨 Docker Build

Creates an image from a Dockerfile.

bash

CopyEdit

docker build -t student-api-image .

---

5. 🚀 Docker Run

Starts a container from an image.

bash

CopyEdit

docker run -d -p 8080:8080 student-api-image

- -d → detached mode
- -p → port mapping (host:container)

---

6. 🧰 Docker Compose

- A tool to run **multiple containers** (e.g., app + DB) using a docker-compose.yml.

📄 Sample docker-compose.yml for Java + MySQL:

yaml

CopyEdit

```
version: "3"

services:

  mysql:

    image: mysql:8

    environment:

      MYSQL_ROOT_PASSWORD: root

      MYSQL_DATABASE: studentdb

    ports:

      - "3306:3306"


  app:

    build: .

    ports:

      - "8080:8080"

    depends_on:

      - mysql
```

📦 Run with:

bash

CopyEdit

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

📂 Common Docker Commands

Command	Description
docker ps	Show running containers
docker images	List all images
docker build -t image-name .	Build image from Dockerfile
docker run -p 8080:8080 image-name	Run container
docker stop <container_id>	Stop a running container
docker rm <container_id>	Remove a container
docker rmi <image_id>	Remove an image

🛡️ Benefits for Java Developers

Benefit	Description
✓ Microservices Ready	Run multiple services (Spring Boot, Redis, etc.)
✓ Clean Environment	No conflict between versions
✓ Easy DB Setup	Run MySQL, MongoDB with 1 command
✓ Great with CI/CD	Used in Jenkins, GitHub Actions, etc.

🚫 Drawbacks

Drawback	Detail
❌ Learning Curve	Initial setup takes time
❌ Performance	Slight overhead on Windows/Mac
❌ Debugging	Harder than native dev setup

✅ Optional but Important Docker Topics

🔹 1. Docker Volumes (Persistent Data Storage)

📌 What is a Volume?

A **Docker Volume** is a **way to store data outside the container**, so that data is **not lost** when the container stops or is deleted.

🧠 Why Use It?

- Containers are **ephemeral** — when removed, all data is gone.
- Volumes help **persist data** (like database data, logs, etc.)
- Easier backup and sharing between containers.

📦 Real-Life Example:

You're running a MySQL container. Without volume, all your data is lost when container stops.
With volume:

bash

CopyEdit

```
docker run -d -v mysql_data:/var/lib/mysql mysql
```

Here, mysql_data is the **named volume** storing your DB data.

🔧 Volume Commands:

Command	Purpose
docker volume create volname	Creates a volume
docker volume ls	Lists all volumes
docker volume inspect volname	Shows details
docker volume rm volname	Deletes volume

🔹 2. Docker Networks (Communication Between Containers)

📌 What is a Docker Network?

Docker network allows containers to **communicate with each other** securely and efficiently.

🧠 Why Use It?

- Microservices (e.g., app → DB → Redis) need to talk to each other.
- Use **user-defined bridge networks** to enable name-based access (no IP).

🔁 Real-Life Example:

Running a Spring Boot app that needs MySQL:

bash

CopyEdit

```
docker network create mynet

docker run -d --name mysql --network mynet mysql

docker run -d --name app --network mynet myapp
```

Now the app container can connect to DB using:

java

CopyEdit

```
jdbc:mysql://mysql:3306/studentdb
```

(No IP needed! Just use the container name as hostname.)

🔧 Network Commands:

Command	Description
docker network create mynet	Create network
docker network ls	List networks
docker network inspect mynet	View details

Command	Description
docker network rm mynet	Remove network

🔹 3. Docker Registry (Image Sharing)

📌 What is Docker Registry?

A **central place to store and share Docker images**.
Public: Docker Hub
Private: You can create your own registry

🧠 Why Use It?

- To **share images** across teams or servers
- To **deploy** from CI/CD pipelines (Jenkins, GitHub Actions)

🌐 Real-Life Use:

You build a Docker image of your Spring Boot app and push it to Docker Hub:

bash

CopyEdit

docker tag myapp username/myapp:1.0

docker login

docker push username/myapp:1.0

Others can pull it using:

bash

CopyEdit

docker pull username/myapp:1.0

🔧 Registry Commands:

Command	Purpose
docker login	Authenticate to Docker Hub
docker tag	Rename an image
docker push	Upload to registry
docker pull	Download from registry

✅ Summary Table: Should You Learn?

Feature	Use It If...	Required for Interview?
🔹 Volumes	App stores data (DB, logs, uploads)	✅ Yes
🔹 Networks	Multiple containers need to talk	✅ Yes (basic level)
🔹 Docker Registry	You need to share/deploy Docker images	✅ Yes