Docker

✅ STEP 1: Docker + Docker Compose (Start Now)

We will do it properly, step-by-step.

✅ 1. What services we will dockerize?

You have these services:

1. eureka-service-registry (8761)
2. api-gateway (8080)
3. user-service (8083)
4. product-service (8081)
5. cart-service (8082)
6. order-service (8084)
7. payment-service (8085)
8. postgres database (5432)

✅ 2. What you need to do FIRST (Before Dockerfile)

🔥 **Change all service Eureka URLs**

Because in docker, localhost won't work.

In every service (user/product/cart/order/payment/api-gateway):

Replace:

defaultZone: http://localhost:8761/eureka/

With:

defaultZone: http://eureka-server:8761/eureka/

✅ 3. Update DB URL in each service

Replace:

spring.datasource.url=jdbc:postgresql://localhost:5432/advann_db

With:

spring.datasource.url=jdbc:postgresql://postgres-db:5432/advann_db

✅ 4. Create Dockerfile for each service

Inside each microservice root folder, create:

📌 Dockerfile

Standard Dockerfile (Spring Boot JAR)

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","app.jar"]
```

⚠️ For each service EXPOSE will be different:

- eureka 8761
- gateway 8080
- user 8083
  etc.

✅ 5. Create docker-compose.yml (Main File)

We will create one compose file that starts:

✅ postgres-db
✅ eureka-server
✅ api-gateway
✅ all microservices

Create docker-compose.yml (Main File)

Create in your main project folder:

```
version: "3.8"

services:

  postgres-db:

    image: postgres:15

    container_name: postgres-db

    restart: always

    environment:

      POSTGRES_DB: advann_db

      POSTGRES_USER: postgres

      POSTGRES_PASSWORD: root

    ports:

      - "5432:5432"

  eureka-server:

    build: ./eureka-service-registry

    container_name: eureka-server

    restart: always

    ports:

      - "8761:8761"

    depends_on:

      - postgres-db

  api-gateway:

    build: ./api-gateway

    container_name: api-gateway

    restart: always
    ports:
      - "8080:8080"
    depends_on:
      - eureka-server
  user-service:
    build: ./user-service
    container_name: user-service
    restart: always
    ports:
      - "8083:8083"
    depends_on:
      - eureka-server
      - postgres-db

  product-service:
    build: ./product-service
    container_name: product-service
    restart: always
    ports:
      - "8081:8081"
    depends_on:
      - eureka-server
      - postgres-db

  cart-service:
    build: ./cart-service
    container_name: cart-service
```

```yaml
  restart: always

  ports:

    - "8082:8082"

  depends_on:

    - eureka-server

    - postgres-db


order-service:

  build: ./order-service

  container_name: order-service

  restart: always

  ports:

    - "8084:8084"

  depends_on:

    - eureka-server

    - postgres-db


payment-service:

  build: ./payment-service

  container_name: payment-service

  restart: always

  ports:

    - "8085:8085"

  depends_on:

    - eureka-server

    - postgres-db
```

Now next we have to build jar files for each service:

Go inside each service and run:

mvn clean package -DskipTests

Do it for all services.

Important note while running this command if terminal shows error like mvn not recognised that means in your system there maven is not installed so first install maven then add the bin path in environment variable.

✅ **Step 1: Download Maven**

1. Open this site:
   https://maven.apache.org/download.cgi
2. Download:
   Binary zip archive
   Example: apache-maven-3.9.x-bin.zip

✅ **Step 2: Extract Maven**

Extract the zip to a folder like:

📌 C:\Program Files\Apache\maven\

After extracting, it should look like:

📌 C:\Program Files\Apache\maven\apache-maven-3.9.x\

Inside it you will see:

- bin
- conf
- lib

✅ **Step 3: Set MAVEN_HOME Environment Variable**

1. Press **Windows + S**
2. Search: **Environment Variables**
3. Open: **Edit the system environment variables**
4. Click: **Environment Variables**

Now lets start with building jar for each service: After executing the command **mvn clean package -DskipTests**

In all service follow the next step.

✅ **STEP 1F: Run Full Project**

From root folder (where docker-compose.yml exists):

docker-compose up --build

<u>Important note while running this command if terminal shows error like docker not recognised that means in your system there docker is not installed so first install docker.</u>

After installing docker again run the command from the product root folder: **docker-compose up –build**

Now very important thing if suppose you have made mistake or want to update anything in one service application.yml file then that time what will you do:

- Suppose one case you have updated something in payment-service
    - ➢ Open terminal and redirect to that payment-service folder like cd payment-service
    - ➢ When you are inside that payment-service folder then again run the command
    - ➢ **mvn clean package -DskipTests**
    - ➢ After then you have to come out from that folder like cd ..
    - ➢ Rebuild payment-service docker image with no cache
    - ➢ **docker-compose build --no-cache payment-service**
    - ➢ Start again
    - ➢ **docker-compose up**

Now some important things to know about docker:

1. Why we need docker what is the use of docker?
   Ans: Docker is used to containerize applications. It packages a microservice along with all its dependencies into a Docker image and runs it as a container. In microservices architecture, Docker helps us run multiple services easily in any system or environment without dependency issues, making deployment and project setup faster and consistent.

2. How to push the docker images in docker hub?
   Ans: We have to follow some steps:
    - ➢ Open cmd or windows PowerShell and run the command: **docker login**
    - ➢ After you logged in successfully you will get confirmation
    - ➢ Now suppose you have services like user-service, product-service
    - ➢ First, we need to tag the images for that run the command
    - ➢ **docker tag backend-user-service sandipm9903/backend-user-service:latest**
    - ➢ **docker tag backend-product-service sandipm9903/backend-product-service:latest**
    - ➢ After getting tagged we have to push the images
    - ➢ **docker push sandipm9903/backend-user-service:latest**

- docker push sandipm9903/backend-product-service:latest
- In your docker console you will get to see that all your images have been uploaded in your docker hub.

3. Now I have pushed the images now how other will get those images and run my application into their system?

   Ans: They can manually take the pull from Docker hub by executing the command

   ```
   docker pull sandipm9903/backend-user-service:latest
   docker push sandipm9903/backend-product-service:latest
   ```

   Or

   We can do easily I will send my docker-compose file with some minor update which is:

```
version: "3.8"

services:

  postgres-db:
    image: postgres:15
    container_name: postgres-db
    restart: always
    environment:
      POSTGRES_DB: advann_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
    ports:
      - "5432:5432"

  eureka-server:
    image: sandipm9903/backend-eureka-server:latest
    container_name: eureka-server
    restart: always
    ports:
      - "8761:8761"
    depends_on:
      - postgres-db

  user-service:
    image: sandipm9903/backend-user-service:latest
    container_name: user-service
    restart: always
    ports:
      - "8083:8083"
    depends_on:
      - eureka-server
      - postgres-db
```

```yaml
  product-service:
    image: sandipm9903/backend-product-service:latest
    container_name: product-service
    restart: always
    ports:
      - "8081:8081"
    depends_on:
      - eureka-server
      - postgres-db
    environment:
      AWS_ACCESS_KEY: ${AWS_ACCESS_KEY}
      AWS_SECRET_KEY: ${AWS_SECRET_KEY}
      AWS_REGION: ${AWS_REGION}
      AWS_BUCKET_NAME: ${AWS_BUCKET_NAME}
```

Now the other person will paste the file in their file explorer and they will run the command
`docker compose up -d`
Docker will automatically pull all your images from DockerHub. With the help of image name.

💧 Important Note (AWS + Razorpay Keys)
They must create .env file in same folder as we are using aws s3 bucket over here:

```
AWS_ACCESS_KEY=xxxx
AWS_SECRET_KEY=xxxx
AWS_REGION=ap-south-1
AWS_BUCKET_NAME=xxxx
```

4. Docker Image vs Docker Container
   Ans: Docker Image
   Docker Image is like a template / blueprint.
   It contains:
   - your application code (jar)
   - dependencies
   - runtime (Java 17)
   - config files
   - OS layer

   📌 Example:
   backend-user-service:latest
   It is not running, just stored.
   👉 Like: APK file (not installed yet)

   ☑ Docker Container
   Docker Container is the running instance of an image.
   When you run an image, it becomes a container.
   📌 Example:
   user-service container running on port 8083
   👉 Like: App installed and running

💧 Easy Example
Image = Class
Container = Object

☑ Interview Answer (Perfect)
"Docker image is a static package that contains application and dependencies. Docker container is the runtime instance of that image. One image can create multiple containers."

☑ Example Command
Create container from image:
docker run backend-user-service
Now container starts.

If interviewer asks further, tell them:
☑ Images are read-only
☑ Containers are running + writable layer

INTERVIEW QUESTIONS AND ANSWERS

☑ 1. What is Docker?

Answer:
Docker is a containerization platform used to package an application along with its dependencies into containers, so it runs consistently across different environments.

☑ 2. What is the difference between Virtual Machine and Docker?

Answer:
A VM runs a full operating system with its own kernel, so it is heavy and slow.
Docker containers share the host OS kernel, so they are lightweight, faster, and consume less resources.

☑ 3. What is Docker Image?

Answer:
A Docker image is a read-only blueprint that contains application code, runtime, dependencies, and configuration required to run the app.

☑ 4. What is Docker Container?

Answer:
A Docker container is the running instance of a Docker image.
An image can create multiple containers.

☑ 5. What is Dockerfile?

Answer:
Dockerfile is a script containing instructions to build a Docker image, such as base image, copy files, install dependencies, expose port, and run commands.

☑ 6. What is the use of docker-compose.yml?

Answer:
Docker Compose is used to run multiple containers (microservices + DB + gateway) together using a single configuration file and a single command.

Example:

```
docker compose up -d
```

☑ 7. Difference between docker build and docker run?

Answer:

- docker build creates an image from Dockerfile.
- docker run starts a container from that image.

☑ 8. What is the difference between COPY and ADD in Dockerfile?

Answer:

- COPY only copies files from local to container.
- ADD can also extract tar files and download from URL.

In industry we prefer COPY.

☑ 9. What is the difference between CMD and ENTRYPOINT?

Answer:

- ENTRYPOINT defines the main command that always runs.
- CMD provides default arguments and can be overridden.

Mostly Spring Boot uses:

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

☑ 10. What is Docker volume?

Answer:
A Docker volume is used to persist data outside the container. Because container data gets deleted when container stops.

Example:
Postgres data should be stored in volume.

☑ **11. What happens if you delete a container?**

Answer:
Container data will be lost unless it is stored in a Docker volume. That's why DB containers always use volumes.

☑ **12. What is the use of DockerHub?**

Answer:
DockerHub is a container registry where we push and store Docker images so that servers or other developers can pull and run them without source code.

☑ **13. How do you push Docker image to DockerHub?**

Answer:
Steps:

1. docker login
2. docker tag image username/image:tag
3. docker push username/image:tag

☑ **14. How will someone run your project without source code?**

Answer:
If the Docker images are pushed to DockerHub, then using docker-compose with image: field, anyone can run the entire system with:

docker compose up -d

Docker will automatically pull images from DockerHub.

☑ **15. What is the difference between build: and image: in docker-compose?**

Answer:

- build: builds image locally using source code.
- image: pulls image from DockerHub registry.

For production, we use image:.

☑ 16. What is the use of depends_on in docker-compose?

Answer:
It ensures startup order. Example: user-service should start after postgres and eureka.

But it does not guarantee DB is fully ready, it only ensures container started.

☑ 17. How do containers communicate in Docker Compose?

Answer:
Docker Compose creates a default network, and services can communicate using their service name.

Example:

jdbc:postgresql://postgres-db:5432/advann_db

☑ 18. Why localhost doesn't work inside Docker container?

Answer:
Inside a container, localhost refers to the container itself, not the host machine.
So we use container/service names like postgres-db, eureka-server.

☑ 19. How do you check running containers?

Answer:

docker ps

☑ 20. How do you check logs of a container?

Answer:

docker logs <container_name>

Example:

docker logs api-gateway

🔥 Most Important Interview One-Liner (Use This)

"Docker helps in consistent deployment by packaging application and runtime into containers. Docker Compose is used for microservices to run multiple services together. DockerHub is used to store and share images."

🐳 Docker Basic Commands

✅ Check Docker version

```
docker --version
```

✅ Login to DockerHub

```
docker login
```

📦 Image Related Commands

✅ Build image from Dockerfile

```
docker build -t image-name
```

Example:

```
docker build -t backend-user-service
```

✅ List all images

```
docker images
```

✅ Tag image for DockerHub

```
docker tag image-name username/image-name:latest
```

Example:

```
docker tag backend-user-service sandipm9903/backend-user-service:latest
```

✅ Push image to DockerHub

```
docker push username/image-name:latest
```

✅ Pull image from DockerHub

```
docker pull username/image-name:latest
```

✅ Remove image

```
docker rmi image-name
```

🚀 Container Related Commands

✅ Run container

```
docker run image-name
```

With port mapping:

```
docker run -p 8083:8083 backend-user-service
```

✅ Run container in background

```
docker run -d image-name
```

✅ List running containers

```
docker ps
```

✅ List all containers (including stopped)

```
docker ps -a
```

✅ Stop container

```
docker stop container-name
```

✅ Start stopped container

```
docker start container-name
```

✅ Remove container

```
docker rm container-name
```

✅ View container logs

```
docker logs container-name
```

Example:

```
docker logs api-gateway
```

🧱 Docker Compose Commands

✅ Start all services

```
docker compose up
```

✅ **Start in background**

```
docker compose up -d
```

✅ **Stop all services**

```
docker compose down
```

✅ **Rebuild images**

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

✅ **View compose logs**

```
docker compose logs
```

💾 **Volume Commands**

✅ **List volumes**

```
docker volume ls
```

✅ **Remove volume**

```
docker volume rm volume-name
```

🌐 **Network Commands**

✅ **List networks**

```
docker network ls
```

🧠 **Important Real-World Debug Commands**

Enter inside running container

```
docker exec -it container-name /bin/bash
```

Example:

```
docker exec -it user-service /bin/bash
```

🔥 **Most Common Errors + Fix**

❌ **Port already in use**

Check:

```
docker ps
```

Stop container using same port.

❌ Image not found

Make sure:

```
docker images
```

Or run:

```
docker pull image-name
```
