Dockerization of Microservices - Practice Notes

✧ Dockerfile - Currency Exchange Microservice

Dockerfile

```
FROM maven:3.8.2-jdk-8-slim AS build
WORKDIR /home/app
COPY . /home/app
RUN mvn -f /home/app/pom.xml clean package

FROM eclipse-temurin:8-jdk-alpine
VOLUME /tmp
EXPOSE 8000
COPY --from=build /home/app/target/*.jar app.jar
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -
Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
```

❖ Explanation

- **Stage 1 (Build Stage)**
  - `FROM maven:3.8.2-jdk-8-slim AS build`: Uses a Maven image with JDK 8 to compile the project.
  - `WORKDIR /home/app`: Sets working directory inside the container.
  - `COPY . /home/app`: Copies source code into the container.
  - `RUN mvn -f /home/app/pom.xml clean package`: Builds the JAR file using Maven.

- **Stage 2 (Runtime Stage)**
  - `FROM eclipse-temurin:8-jdk-alpine`: Lightweight JDK runtime image.
  - `VOLUME /tmp`: Defines a temporary volume for file storage (needed by Spring Boot).
  - `EXPOSE 8000`: Exposes port 8000 for external access.
  - `COPY --from=build /home/app/target/*.jar app.jar`: Copies the built JAR from stage 1.
  - `ENTRYPOINT`: Runs the JAR with Java, allowing optional JVM options via `$JAVA_OPTS`.

✧ Dockerfile - Currency Conversion Microservice

Dockerfile

```
FROM maven:3.8.2-jdk-8-slim AS build
WORKDIR /home/app
COPY . /home/app
RUN mvn -f /home/app/pom.xml clean package

FROM eclipse-temurin:8-jdk-alpine
VOLUME /tmp
EXPOSE 8100
COPY --from=build /home/app/target/*.jar app.jar
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -
Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
```

❖ Explanation

- **Build Stage**: Same as currency exchange, compiles the JAR.
- **Runtime Stage**:
  - Exposes **port 8100** (different from exchange service).
  - Everything else mirrors the exchange service setup.

Purpose: Runs the conversion microservice, which depends on the exchange service for data.

1. Building Microservice Images

❖ Currency Exchange Microservice

- **Command:**
- Bash

```
docker build -t souravdevopsdev/currency-exchange:0.0.1 .
```

Process:

- Used **multi-stage build** with Maven (to compile/package) and Eclipse Temurin JDK (to run).
- Maven built the JAR (`mvn clean package`).
- Final stage copied the JAR into a lightweight runtime image.

- **Result:**

  Image tagged as `souravdevopsdev/currency-exchange:0.0.1`

- Currency Conversion Microservice

  - **Command:**
  - Bash

  ```
  docker build -t souravdevopsdev/currency-conversion:0.0.1 .
  ```

  Similar multi-stage build process.

  - **Result:**

    Image tagged as `souravdevopsdev/currency-conversion:0.0.1`

2. Running Containers

Currency Exchange:

Bash

```
docker run -d -p 8000:8000 --name=currency-exchange
souravdevopsdev/currency-exchange:0.0.1
```

Currency Conversion:

Bash

```
docker run -d -p 8100:8100 --name=currency-conversion
souravdevopsdev/currency-conversion:0.0.1
```

Both containers started successfully and were mapped to host ports **`8000` and `8100`**.

3. Networking Challenge

Initially, both services were running on the **default bridge network**.

Problem: Containers on the default bridge cannot resolve each other by name.

The conversion service couldn't reach the **exchange service** using its container name.

Solution Attempt: Tried direct communication, but failed.

4. Fix - Linking & Environment Variable

- Stopped and removed the conversion container:
- Bash

```
docker container stop <id>
```

```
docker container rm <id>
```

Relaunched with **linking** and **environment variable**:

- Bash

```
docker run -d -p 8100:8100
  --env CURRENCY_EXCHANGE_SERVICE_HOST=http://currency-exchange
   --name=currency-conversion  --link currency-exchange
   souravdevopsdev/currency-conversion:0.0.1
```

Why Linking?

- **`--link`** allows one container to reference another **by name**.
- It injects host entries and environment variables so the conversion service can resolve `currency-exchange`.

Why Environment Variable?

The application (Spring Boot + Feign client) needs to know the **base URL** of the exchange service.

- By setting:
- Bash

```
CURRENCY_EXCHANGE_SERVICE_HOST=http://currency-exchange
```

the conversion service could dynamically read the host and connect properly.

5. Logs Verification

- Checked logs with:
- Bash

```
docker logs -f <conversion-container-id>
```

Observed:

i. Spring Boot initialization.
ii. Environment variables loaded, including `CURRENCY_EXCHANGE_SERVICE_HOST`.
iii. Application started successfully on port `8100`.

6. Key Learnings

- **Multi-stage builds** keep images lean and production-ready.
- **Default bridge network limitation**: containers cannot talk by name.
- **Linking** (legacy approach) solves name resolution but is not recommended for modern setups.
- **Environment variables** are essential for service discovery/configuration in microservices.

Best practice: Instead of `--link`, use **user-defined bridge networks** or orchestration tools (Docker Compose, Kubernetes) for scalable service communication.