**✅ Standard Operating Procedure (SOP) for Building and Pushing Docker Images to Artifactory or AWS ECR**

This SOP guides you step-by-step through creating a Docker container for a full-stack application and pushing the image to **JFrog Artifactory** or **AWS Elastic Container Registry (ECR)**.

**📌 Objective**

Build and deploy a containerized full-stack application using:

* **Frontend:** NodeJS 22, React 19, NextJS, Vite, Gatsby.
* **Backend:** Java 21, Spring Boot, MySQL Connector.
* **Database:** MySQL 8.0.

The final Docker image will be pushed to either **JFrog Artifactory** or **AWS ECR**.

**📂 1. Project Directory Structure**

css

CopyEdit

/fullstack-app

│

├── backend/

│ ├── src/

│ ├── pom.xml

│

├── frontend/

│ ├── src/

│ ├── package.json

│

├── Dockerfile

├── docker-compose.yml

├── .dockerignore

**🔧 2. Set Up Dependencies**

**🔵 Backend (pom.xml) Configuration**

Inside the backend/ folder, create a pom.xml with the following dependencies:

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>backend</artifactId>

<version>1.0.0</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.4.1</version>

</parent>

<dependencies>

<dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-data-jpa</artifactId></dependency>

<dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-web</artifactId></dependency>

<dependency><groupId>mysql</groupId><artifactId>mysql-connector-j</artifactId><version>8.0.29</version></dependency>

<dependency><groupId>org.projectlombok</groupId><artifactId>lombok</artifactId></dependency>

<dependency><groupId>io.jsonwebtoken</groupId><artifactId>jjwt-api</artifactId><version>0.12.6</version></dependency>

</dependencies>

</project>

**🔵 Frontend (package.json) Configuration**

In the frontend/ folder, create a package.json:

json

CopyEdit

{

"dependencies": {

"react": "19.0.0",

"jwt-decode": "1.0.2",

"react-router-dom": "7.1.3",

"react-hot-toast": "2.5.1",

"@mui/material": "6.4.2",

"moment": "2.30.1",

"axios": "1.7.9",

"react-icons": "1.0.0",

"react-hook-form": "7.54.2",

"next": "15.1.6",

"vite": "5.0.0",

"gatsby": "5.11.0"

}

}

Run:

cd frontend

npm install

**🐳 3. Create the Dockerfile**

Place this Dockerfile in the project root:

# Stage 1: MySQL Database Setup

FROM mysql:8.0 AS mysql-db

ENV MYSQL\_ROOT\_PASSWORD=root \

MYSQL\_DATABASE=backenddb \

MYSQL\_USER=backenduser \

MYSQL\_PASSWORD=backendpass

EXPOSE 3306

# Stage 2: Frontend Build using RHEL UBI with NodeJS 22

FROM registry.redhat.io/ubi8/nodejs-22 AS frontend

WORKDIR /app/frontend

COPY frontend/package\*.json ./

RUN npm install

COPY frontend/ .

RUN npm run build

# Stage 3: Backend Build using OpenJDK 21 and Maven 3.9.5

FROM openjdk:21-jdk-slim AS backend

WORKDIR /app/backend

RUN apt-get update && apt-get install -y curl bash \

&& curl -sSL https://downloads.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz | tar -xz -C /opt/

ENV M2\_HOME=/opt/apache-maven-3.9.5

ENV PATH=$M2\_HOME/bin:$PATH

COPY backend/pom.xml .

RUN mvn dependency:resolve

COPY backend/ .

RUN mvn clean package

# Stage 4: Final Application Image

FROM openjdk:21-jdk-slim

WORKDIR /app

COPY --from=backend /app/backend/target/backend-1.0.0.jar ./backend.jar

COPY --from=frontend /app/frontend/.next /app/frontend/.next

ENV SPRING\_DATASOURCE\_URL=jdbc:mysql://mysql-db:3306/backenddb \

SPRING\_DATASOURCE\_USERNAME=backenduser \

SPRING\_DATASOURCE\_PASSWORD=backendpass

EXPOSE 3306 8080 3000

CMD ["sh", "-c", "java -jar backend.jar & npm run start --prefix /app/frontend"]

**📋 4. Create Docker Compose File**

Create a docker-compose.yml:

version: '3.8'

services:

mysql:

image: mysql:8.0

container\_name: mysql-db

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: backenddb

MYSQL\_USER: backenduser

MYSQL\_PASSWORD: backendpass

ports:

- "3306:3306"

app:

build: .

container\_name: fullstack-app

ports:

- "3000:3000"

- "8080:8080"

depends\_on:

- mysql

**🔨 5. Build and Tag Docker Image**

Run the following commands from the project root:

docker-compose build

**🚀 6. Push Docker Image to Artifactory or AWS ECR**

**🔗 A. Push to JFrog Artifactory**

1. **Login:**

docker login your-artifactory-url -u your-username -p your-password

1. **Tag and Push:**

docker tag fullstack-app:latest your-artifactory-url/repo-name/fullstack-app:latest

docker push your-artifactory-url/repo-name/fullstack-app:latest

**☁️ B. Push to AWS ECR**

1. **Authenticate:**

aws ecr get-login-password --region your-region | docker login --username AWS --password-stdin your-aws-account-id.dkr.ecr.your-region.amazonaws.com

1. **Create ECR Repository:**

aws ecr create-repository --repository-name fullstack-app

1. **Tag and Push:**

bash

CopyEdit

docker tag fullstack-app:latest your-aws-account-id.dkr.ecr.your-region.amazonaws.com/fullstack-app:latest

docker push your-aws-account-id.dkr.ecr.your-region.amazonaws.com/fullstack-app:latest

**✅ 7. Run the Docker Container**

docker-compose up -d

**🔍 8. Verify the Deployment**

* **MySQL:** Connect using:

docker exec -it mysql-db mysql -u backenduser -p

* **Backend (Spring Boot):** Check:

http://localhost:8080

* **Frontend (React, Next.js):** Open:

http://localhost:3000