Índice

1. Introducción

1.1 Descripción del Proyecto

Este proyecto consiste en la creación de una **plataforma de subastas en línea** utilizando una arquitectura de microservicios. Los servicios incluyen un **backend**, un **frontend**, y una **base de datos**. Para gestionar y desplegar estos servicios de manera eficiente, se utiliza **Docker** en la solución local y **Kubernetes** para la orquestación en entornos de producción, como la nube.

1.2 Objetivos del Proyecto

Los objetivos principales de este proyecto incluyen:

- Desarrollar una aplicación funcional para realizar subastas en línea.
- Contenerizar los servicios mediante Docker para facilitar la ejecución y escalabilidad.
- Implementar un sistema de despliegue automático usando Kubernetes para una mejor gestión de recursos y escalabilidad en la nube.
- Garantizar la persistencia de datos y la interacción fluida entre los diferentes servicios (base de datos, backend y frontend).

2. Solución Local Docker

2.1 Dockerfile Backend

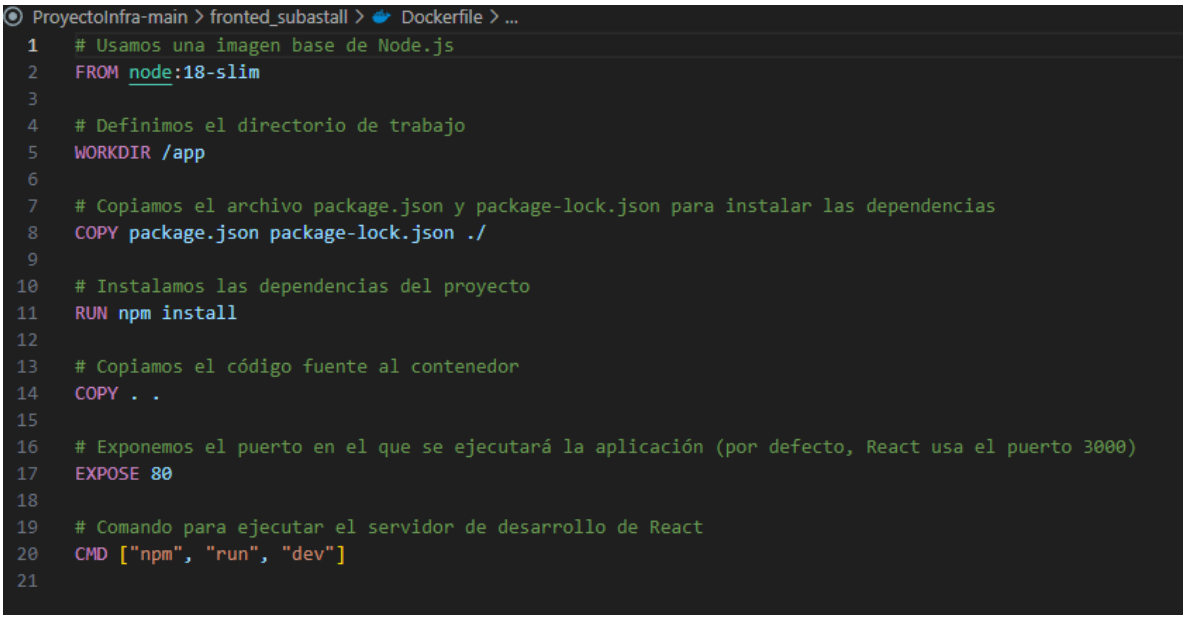
El **Dockerfile** para el backend define el contenedor para la aplicación. Incluye las configuraciones para la construcción de la imagen, como la instalación de dependencias, la configuración de puertos y la ejecución de la aplicación. El contenedor está basado en una imagen de Java para ejecutar el backend desarrollado en Spring Boot.

ProyectoInfra-main > SpringSubastAll > Dockerfile > ...

```dockerfile
# Usamos una imagen base de Maven que ya tiene Java instalado
FROM maven:3.9.0-eclipse-temurin-17 as build

# Definimos el directorio de trabajo
WORKDIR /app

# Copiamos el archivo pom.xml y el código fuente para que Maven lo compile
COPY pom.xml .
COPY src ./src

# Ejecutamos Maven para compilar el proyecto y empaquetarlo en un JAR
RUN mvn clean package -DskipTests

# Usamos una imagen base de OpenJDK para ejecutar la aplicación
FROM openjdk:17-jdk-slim

# Definimos el directorio de trabajo en el contenedor
WORKDIR /app

# Copiamos el archivo JAR generado desde la etapa de construcción
COPY --from=build /app/target/demo-jwt-0.0.1-SNAPSHOT.jar app.jar

# Copiamos la carpeta "uploads" al contenedor
COPY uploads /app/uploads

# Exponemos el puerto 8080
EXPOSE 8080

# Comando para ejecutar la aplicación
ENTRYPOINT ["java", "-jar", "app.jar"]

```

2.2 Dockerfile Frontend

El **Dockerfile** para el frontend está diseñado para contenerizar la aplicación web que interactúa con el backend. En este caso, se utiliza una imagen base de **Node.js** para construir la aplicación y luego se sirve a través de **nginx**.



2.3 docker-compose.yml

El archivo docker-compose.yml permite definir y gestionar los contenedores de todos los servicios, como la base de datos, el backend y el frontend. Aquí se configura la red, volúmenes y puertos para que los contenedores puedan comunicarse entre sí.

ProyectoInfra-main > docker-compose.yml

```yaml
services:
  database:
    image: postgres:latest
    environment:
      POSTGRES_USER: administrador
      POSTGRES_PASSWORD: admin
      POSTGRES_DB: subasta
    networks:
      - backend
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  back:
    image: subasta-back
    environment:
      DB_HOST: database
      DB_PORT: 5432
      DB_NAME: subasta
      DB_USER: administrador
      DB_PASSWORD: admin
    networks:
      - backend
    ports:
      - "8080:8080"
    volumes:
      - fotos-data:/app/uploads/fotos
    depends_on:
      - database

  frontend:
    image: subasta-frontend
    environment:
      API_URL: 'http://back:8080'
    networks:
      - backend
    ports:
      - "80:80"
    depends_on:
      - back

networks:
  backend:
    driver: overlay

volumes:
```

2.4 Imágenes Montadas

En esta sección, se especifica que las imágenes de Docker construidas para el backend y el frontend son montadas en los contenedores correspondientes. Además, se asegura la persistencia de los datos en volúmenes para la base de datos.