

Índice

1. **Introducción**
 - 1.1 Descripción del Proyecto
 - 1.2 Objetivos del Proyecto
2. **Solución Local Docker**
 - 2.1 Dockerfile Backend
 - 2.2 Dockerfile Frontend
 - 2.3 docker-compose.yml
 - 2.4 Imágenes Montadas
3. **Solución en la Nube**
 - 4.1 Creación de Clúster de Kubernetes en Google Cloud con Autoscaling
4. **Capturas de Funcionamiento de la Aplicación**
5. **Análisis y Conclusiones**
6. **Conclusiones**

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 > SpringSubstAll > Dockerfile > ...
1  # Usamos una imagen base de Maven que ya tiene Java instalado
2  FROM maven:3.9.0-eclipse-temurin-17 as build
3
4  # Definimos el directorio de trabajo
5  WORKDIR /app
6
7  # Copiamos el archivo pom.xml y el código fuente para que Maven lo compile
8  COPY pom.xml .
9  COPY src ./src
10
11 # Ejecutamos Maven para compilar el proyecto y empaquetarlo en un JAR
12 RUN mvn clean package -DskipTests
13
14 # Usamos una imagen base de OpenJDK para ejecutar la aplicación
15 FROM openjdk:17-jdk-slim
16
17 # Definimos el directorio de trabajo en el contenedor
18 WORKDIR /app
19
20 # Copiamos el archivo JAR generado desde la etapa de construcción
21 COPY --from=build /app/target/demo-jwt-0.0.1-SNAPSHOT.jar app.jar
22
23 # Copiamos la carpeta "uploads" al contenedor
24 COPY uploads /app/uploads
25
26 # Exponemos el puerto 8080
27 EXPOSE 8080
28
29 # Comando para ejecutar la aplicación
30 ENTRYPOINT ["java", "-jar", "app.jar"]
31
```

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**.

```
ProyectoInfra-main > fronted_subastall > Dockerfile > ...
1  # Usamos una imagen base de Node.js
2  FROM node:18-slim
3
4  # Definimos el directorio de trabajo
5  WORKDIR /app
6
7  # Copiamos el archivo package.json y package-lock.json para instalar las dependencias
8  COPY package.json package-lock.json ./
9
10 # Instalamos las dependencias del proyecto
11 RUN npm install
12
13 # Copiamos el código fuente al contenedor
14 COPY . .
15
16 # Exponemos el puerto en el que se ejecutará la aplicación (por defecto, React usa el puerto 3000)
17 EXPOSE 80
18
19 # Comando para ejecutar el servidor de desarrollo de React
20 CMD ["npm", "run", "dev"]
21
```

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

```
1  services:
2    database:
3      image: postgres:latest
4      environment:
5        POSTGRES_USER: administrador
6        POSTGRES_PASSWORD: admin
7        POSTGRES_DB: subasta
8      networks:
9        - backend
10     volumes:
11       - db-data:/var/lib/postgresql/data
12     ports:
13       - "5432:5432"
14
15     back:
16       image: subasta-back
17       environment:
18         DB_HOST: database
19         DB_PORT: 5432
20         DB_NAME: subasta
21         DB_USER: administrador
22         DB_PASSWORD: admin
23       networks:
24         - backend
25       ports:
26         - "8080:8080"
27       volumes:
28         - fotos-data:/app/uploads/fotos
29       depends_on:
30         - database
31
32     frontend:
33       image: subasta-frontend
34       environment:
35         API_URL: 'http://back:8080'
36       networks:
37         - backend
38       ports:
39         - "80:80"
40       depends_on:
41         - back
42
43     networks:
44       backend:
45         driver: overlay
46
47     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.