MEMORIA TÉCNICA

Orquestación Eficiente de Servicios: Implementación de un Sistema CRUD Utilizando Docker

Participante: Luis González

Participante: Abelardo Leon

Introducción

Este informe aborda el proceso de creación y ejecución de una aplicación CRUD (Crear, Leer, Actualizar, Eliminar) mediante el uso eficiente de Docker para coordinar los servicios tanto del backend como del frontend. La aplicación, diseñada con el objetivo de gestionar datos de manera óptima, fusiona tecnologías como Python (Flask) para el backend, Angular para el frontend y Docker para encapsular eficientemente los servicios, incluso la creación de una base de datos SQL.

En un panorama tecnológico en constante cambio, la implementación de soluciones efectivas para el desarrollo y despliegue de aplicaciones se torna fundamental. En este contexto, la orquestación de servicios a través de contenedores emerge como una solución sólida y versátil.

Objetivos del Proyecto:

1. Desarrollar un backend utilizando Python y el framework Flask para proporcionar las operaciones CRUD a la base de datos.
2. Crear una base de datos SQL que sirva como motor de almacenamiento para la aplicación.
3. Crear un frontend interactivo utilizando Angular para consumir los servicios proporcionados por el backend.
4. Orquestar eficientemente los servicios backend, base de datos y frontend utilizando Docker Compose.

Estructura del Proyecto:

La estructura del proyecto permite un fácil despliegue y mantenimiento, asegurando la independencia y escalabilidad de cada componente.

La organización del proyecto se ha diseñado cuidadosamente para garantizar la claridad, modularidad y fácil mantenimiento. La estructura del proyecto se compone de tres partes principales: database, BackPython, y FrontAngular.

Carpeta Raíz: app

docker-compose.yml:

Este archivo define la configuración de Docker Compose para orquestar los servicios de la aplicación. Incluye la definición de servicios para la base de datos, backend y frontend, así como la configuración de redes y direcciones IP específicas.

Sub carpeta: database

database.sql:

Contiene el script SQL para la creación de la base de datos y las tablas necesarias. Este script se ejecuta automáticamente durante la inicialización del contenedor de la base de datos.

Dockerfile:

Define la configuración para construir la imagen de Docker del servicio de base de datos. Utiliza la imagen oficial de MySQL y copia el script SQL en el directorio de inicialización.

Sub carpeta: BackPython

back.py:

Contiene la lógica del backend desarrollado en Python utilizando el framework Flask. Proporciona endpoints para realizar operaciones CRUD en la base de datos SQL.

Dockerfile:

Define la configuración para construir la imagen de Docker del servicio backend. Utiliza la imagen oficial de Python y copia el código fuente del backend al directorio de trabajo.

Sub carpeta: FrontAngular

angularProyect:

Contiene el proyecto Angular. Dentro de este directorio, se encuentra la carpeta del proyecto Angular, que a su vez contiene todos los archivos y carpetas generados por Angular CLI.

Dockerfile:

Define la configuración para construir la imagen de Docker del servicio frontend. Utiliza la imagen oficial de Node.js para construir la aplicación Angular y luego la sirve utilizando Nginx.

Detalles del proyecto:

Base de Datos:

Descripción del Esquema de la Base de Datos y su Relación con el Proyecto:

El esquema de la base de datos se compone de varias tablas interrelacionadas para gestionar clientes, productos, vendedores y las ventas asociadas. La estructura incluye tablas como clientes, productos, vendedores y ventas, que están diseñadas para facilitar la gestión integral de datos en el contexto de una aplicación de ventas.

Detalles sobre el Archivo SQL de Inicialización y su Importancia:

El archivo database.sql actúa como el script de inicialización que crea la base de datos y las tablas necesarias. Define las relaciones entre las tablas y proporciona datos de ejemplo para ilustrar el funcionamiento del sistema. Al utilizar este script, se asegura la consistencia en la creación de la base de datos, lo que simplifica el despliegue y la configuración inicial.

Configuración Específica de Docker para el Servicio de Base de Datos:

El servicio de base de datos se ha dockerizado utilizando la imagen oficial de MySQL. El archivo Dockerfile en la carpeta database establece variables de entorno para la configuración de MySQL, copia el script SQL a la carpeta de inicialización (/docker-entrypoint-initdb.d/), y expone el puerto 3306 para permitir la comunicación con otros servicios.

Backend:

Descripción del Backend Desarrollado en Python (Flask):

El backend, implementado en Python con el framework Flask, proporciona un conjunto de API RESTful para realizar operaciones CRUD. Está diseñado para gestionar las solicitudes del frontend y manipular los datos en la base de datos SQL. Flask-CORS se utiliza para gestionar la política de mismo origen, permitiendo la comunicación desde el frontend.

Conexión y Comunicación con la Base de Datos:

La conexión con la base de datos se realiza mediante el conector MySQL para Python. El archivo back.py configura las credenciales y la información de conexión, estableciendo una

conexión cada vez que se realiza una operación CRUD. Las consultas SQL son ejecutadas para interactuar con la base de datos y devolver los resultados al frontend.

Endpoint del CRUD y Cómo Interactúa con la Base de Datos:

El backend expone varios endpoints para realizar operaciones CRUD. Por ejemplo:

GET /ventas: Obtiene todas las ventas.

GET /ventas/<int:venta_id>: Obtiene una venta específica por ID.

POST /ventas: Crea una nueva venta.

PUT /ventas/<int:venta_id>: Actualiza una venta existente.

DELETE /ventas/<int:venta_id>: Elimina una venta por ID.

Cada endpoint se comunica con la base de datos según la operación realizada, garantizando la integridad de los datos.

Frontend:

Descripción del Frontend Desarrollado en Angular:

El frontend, construido con Angular, proporciona una interfaz de usuario interactiva para interactuar con el backend. Está estructurado en componentes como listar y venta, que ofrecen funcionalidades específicas.

Integración con el Backend y Cómo Consume los Datos:

El servicio BackendService facilita la comunicación con el backend. Los componentes utilizan este servicio para realizar solicitudes HTTP y consumir los datos proporcionados por el backend. Por ejemplo, el componente listar utiliza el servicio para obtener la lista de ventas y el componente venta lo emplea para crear nuevas ventas.

Funcionalidades Específicas y Componentes Utilizados:

Componente listar:

- Muestra la lista de ventas en una tabla.
- Permite eliminar ventas existentes.

Componente venta:

- Proporciona un formulario para crear nuevas ventas.

Dockerización:

Detalles sobre el Dockerfile de Cada Servicio:

Database:

Utiliza la imagen oficial de MySQL.

Copia el script SQL a la carpeta de inicialización.

Define variables de entorno para configurar MySQL.

Servicio Backend (Python/Flask):

BackPython:

Utiliza la imagen oficial de Python.

Instala las dependencias necesarias (Flask, mysql-connector-python).

Copia el código fuente del backend al contenedor.

Servicio Frontend (Angular):

AngularProyect:

Utiliza la imagen oficial de Node.js para construir la aplicación Angular.

Copia el proyecto Angular al contenedor.

Utiliza Nginx para servir la aplicación Angular.

Uso de Docker Compose para la Orquestación de Servicios:

El archivo docker-compose.yml define los servicios (database, back, front) y sus configuraciones específicas. Docker Compose permite orquestar la ejecución simultánea de estos servicios, asegurando una comunicación efectiva entre ellos.

Configuración de Redes y Direcciones IP Específicas para la Comunicación entre Servicios:

Se han definido dos redes en Docker Compose: myapp-network y myapp-external. La primera se utiliza para la comunicación interna entre los servicios, y la segunda proporciona una interfaz externa para la conexión del frontend con el backend. Se han asignado direcciones IP específicas a cada servicio para facilitar la comunicación y asegurar una orquestación eficiente.

Despliegue y Ejecución

Requisitos Previos

Antes de iniciar el despliegue y ejecución de la aplicación, asegúrese de tener instalados los siguientes requisitos:

- Docker
- Git

Pasos para Desplegar y Ejecutar la Aplicación

Clonar el Repositorio:

Clone el repositorio de la aplicación desde el repositorio remoto:

- git clone https://github.com/Alfiie1203/CRUD_Docker_Cloud.git

Cambie al directorio del proyecto:

- cd app

Iniciar los Contenedores:

Inicie los contenedores utilizando Docker Compose:

- docker compose up

puede usar el modificador -d para ejecutar los contenedores en segundo plano.

Acceder a la Aplicación Frontend:

Abra un navegador web y vaya a http://localhost. Debería ver la aplicación frontend funcionando.

Acceder a la Interfaz de Administración de MySQL (Opcional):

Si necesita acceder a la interfaz de administración de MySQL, puede utilizar herramientas como phpMyAdmin o Adminer. Conéctese al servidor MySQL utilizando la dirección IP y el puerto definido en el archivo docker-compose.yml para el servicio de base de datos.

Desafíos y Soluciones

Durante el desarrollo, implementación y mantenimiento de un proyecto como el descrito, se pueden enfrentar diversos desafíos. Aquí se presentan algunos de ellos junto con posibles soluciones:

Gestión de Versiones y Colaboración

Descripción:

Coordinar y gestionar eficientemente las diferentes versiones del código, especialmente cuando varios desarrolladores trabajan simultáneamente.

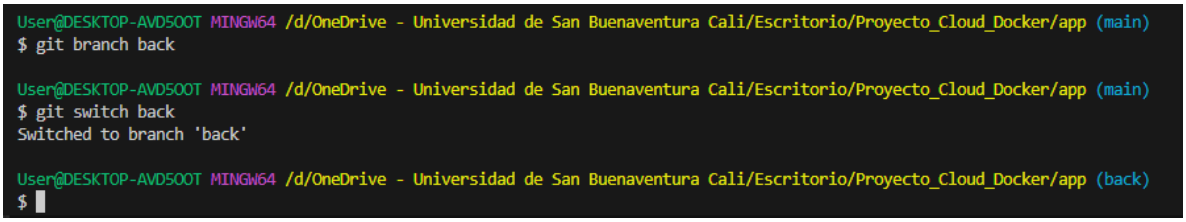
Soluciones Implementadas con GitHub:

Utilización de Git y GitHub:

Se empleó Git como sistema de control de versiones, y GitHub como plataforma de alojamiento remoto para el repositorio.

Estrategia de Ramificación:

Se crearon ramas específicas para el desarrollo del backend y frontend, facilitando el trabajo paralelo y la implementación de nuevas funcionalidades sin afectar la rama principal.



```
User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (back)
$ git add .

User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (back)
$ git commit -m "Agregando el Back"
[back 0b8a22b] Agregando el Back
 2 files changed, 99 insertions(+)
 create mode 100644 backPython/Dockerfile
 create mode 100644 backPython/back.py

User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (back)
$ git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (main)
$ git merge back
Updating d60623b..0b8a22b
Fast-forward
 backPython/Dockerfile |  0
 backPython/back.py    | 99 +++++++++++++++++++++++++++++++++++++++++++++++++++
 2 files changed, 99 insertions(+)
 create mode 100644 backPython/Dockerfile
 create mode 100644 backPython/back.py
```

```
User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.39 KiB | 237.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Alfiie1203/CRUD_Docker_Cloud.git
   d60623b..0b8a22b  main -> main
```

Merges en la Rama Principal "main":

merge backend -> main:

Integración de las nuevas características del backend en la rama principal.

merge frontend -> main:

Incorporación de las mejoras y funciones del frontend en la rama principal.

```
User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (main)
$ git merge front
Already up to date.

User@DESKTOP-AVD5OOT MINGW64 /d/OneDrive - Universidad de San Buenaventura Cali/Escritorio/Proyecto_Cloud_Docker/app (main)
$ git merge back
Already up to date.
```

Resultados y Conclusiones

1. Creación Exitosa de la Base de Datos y CRUD

Resultado:

Se logró implementar una base de datos en MySQL con la estructura definida, y se realizaron operaciones CRUD (Crear, Leer, Actualizar, Eliminar) a través del backend desarrollado en Python con Flask.

Conclusión:

La gestión de datos se llevó a cabo de manera efectiva, permitiendo almacenar y manipular información sobre las ventas.

2. Desarrollo Efectivo del Backend y Frontend

Resultado:

El backend, implementado en Python con Flask, proporciona servicios API RESTful para interactuar con la base de datos. El frontend, desarrollado en Angular, consume estos servicios de manera eficiente.

Conclusión:

La separación clara entre el frontend y el backend facilitó la escalabilidad y el mantenimiento del sistema, permitiendo futuras mejoras y extensiones de funcionalidades.

3. Dockerización y Orquestación de Contenedores

Resultado:

Se logró dockerizar cada componente del proyecto, incluyendo la base de datos, backend y frontend. La orquestación de contenedores mediante Docker Compose facilitó el despliegue y la comunicación entre los servicios.

Conclusión:

La implementación de contenedores Docker mejoró la portabilidad y la consistencia del entorno de desarrollo, simplificando el proceso de despliegue y facilitando la replicación en diferentes entornos.