Desafio 15

Despliegue completo de una aplicación Flask con MySQL, Nginx, Docker Compose y CI/CD

Objetivo

Implementar una aplicación web en Flask con:

- MySQL como base de datos (con réplicas).
- Nginx como balanceador de carga.
- Docker Compose para orquestar los servicios.
- GitHub Actions para CI/CD, exponiendo el servicio mediante Ngrok.

---

1. Configuración de la Aplicación Flask y MySQL

1.1. Aplicación Flask

Reutiliza la aplicación del ejercicio anterior con las siguientes rutas:

- **GET /**: Obtiene todos los registros de la tabla `test`.
- **POST /**: Inserta un nuevo registro en la tabla `test`.

Archivo: `app.py`

```
from flask import Flask, request, jsonify
import mysql.connector
import os

app = Flask(__name__)

def get_db_connection():
    return mysql.connector.connect(
        host=os.getenv('DB_HOST', 'db-master'),
        user=os.getenv('DB_USER', 'root'),
        password=os.getenv('DB_PASSWORD', 'password'),
        database=os.getenv('DB_NAME', 'flaskapp')
    )

@app.route('/', methods=['GET', 'POST'])
def handle_data():
    conn = get_db_connection()
```

```
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
        data = request.json
        cursor.execute("INSERT INTO test (name) VALUES (%s)", (data['name'],))
        conn.commit()
        return jsonify({"message": "Data inserted"}), 201

    cursor.execute("SELECT * FROM test")
    result = cursor.fetchall()
    cursor.close()
    conn.close()
    return jsonify(result)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Archivo SQL para inicializar la base de datos (`init.sql`)

```
CREATE DATABASE flaskapp;
USE flaskapp;
CREATE TABLE test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);
```

1.2. Configuración de MySQL con Réplicas

Configura MySQL como maestro-esclavo:

- El maestro gestiona las escrituras.
- Los esclavos replican los datos para lectura.

Maestro (`db-master`)

Define en `docker-compose.yml`:

```
 db-master:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: password
    volumes:
      - db_master_data:/var/lib/mysql
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "3306:3306"
```

Esclavo (`db-slave`)

Configura para replicar del maestro:

```
 db-slave:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_REPLICATION_USER: repl
      MYSQL_REPLICATION_PASSWORD: password
      MYSQL_MASTER_HOST: db-master
    depends_on:
      - db-master
    volumes:
      - db_slave_data:/var/lib/mysql
```

1.3. Configuración de Nginx como Balanceador de Carga

Define la configuración en `nginx.conf`:

```
upstream flaskapp {
    server flaskapp1:5000;
    server flaskapp2:5000;
}

server {
    listen 80;

    location / {
        proxy_pass http://flaskapp;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

---

2. Crear el Docker Compose

Define los servicios para la aplicación Flask, MySQL maestro-esclavo, y Nginx.

Archivo: `docker-compose.yml`

```
version: '3.8'
```

```
services:
  db-master:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: password
    volumes:
      - db_master_data:/var/lib/mysql
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "3306:3306"

  db-slave:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_REPLICATION_USER: repl
      MYSQL_REPLICATION_PASSWORD: password
      MYSQL_MASTER_HOST: db-master
    depends_on:
      - db-master
    volumes:
      - db_slave_data:/var/lib/mysql

  flaskapp1:
    build:
      context: .
    environment:
      DB_HOST: db-master
      DB_USER: root
      DB_PASSWORD: password
      DB_NAME: flaskapp
    depends_on:
      - db-master

  flaskapp2:
    build:
      context: .
    environment:
      DB_HOST: db-master
      DB_USER: root
      DB_PASSWORD: password
      DB_NAME: flaskapp
    depends_on:
      - db-master

  nginx:
    image: nginx:latest
```

```
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - flaskapp1
      - flaskapp2

volumes:
  db_master_data:
  db_slave_data:
```

---

3. Configurar CI/CD con GitHub Actions

Define un flujo de trabajo para construir y desplegar la aplicación en una VM.

Archivo: `.github/workflows/deploy.yml`

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and push Docker images
        run: |
          docker build -t my-docker-repo/flaskapp:latest .
          docker push my-docker-repo/flaskapp:latest

  deploy:
    runs-on: ubuntu-latest
```

```
  needs: build

  steps:
    - name: SSH to VM and deploy
      uses: appleboy/ssh-action@v0.1.5
      with:
        host: ${{ secrets.VM_HOST }}
        username: ${{ secrets.VM_USER }}
        password: ${{ secrets.VM_PASSWORD }}
        script: |
          cd /path/to/deployment
          docker-compose pull
          docker-compose up -d
```

---

4. Pruebas y Verificación

1. **Accede a la aplicación:**

Usa Ngrok para exponer la aplicación:
ngrok http 80

   -
   - Prueba los endpoints:
     - **POST:** Inserta datos.
     - **GET:** Obtiene datos.
2. **Monitorea CI/CD:**

   - Realiza un push a la rama `main` y verifica la ejecución en GitHub Actions.
