UNIVERSIDAD NACIONAL DEL ALTIPLANO

Facultad de Ingeniería Estadística e Informática

Escuela Profesional de Ingeniería Estadística e Informática

Curso: Sistemas Distribuidos Concurrentes

Presenta:

Paredes Condori Edaly Yamileth

Docente:

Torres Cruz Fred

2024 - II

Configuración de un Sistema Distribuido con Docker Compose y PostgreSQL

December 19, 2024

Crea un archivo `docker-compose.yml`

El archivo `docker-compose.yml` describe los servicios de tu sistema distribuido. Este archivo será el núcleo del proyecto.

1. Abre VS Code y crea un nuevo archivo llamado `docker-compose.yml` en la carpeta de trabajo.

```
version: '3.9'
services:
  db_leader:
    image: postgres:15
    container_name: db_leader
    environment:
      POSTGRES_USER: leader_user
      POSTGRES_PASSWORD: leader_pass
      POSTGRES_DB: leader_db
    volumes:
      - db_leader_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    command: >
      postgres -c wal_level=replica \
               -c max_wal_senders=5 \
               -c wal_keep_size=64MB \
               -c archive_mode=on \
               -c archive_command='/bin/true'

  db_follower_1:
    image: postgres:15
    container_name: db_follower_1
    environment:
      POSTGRES_USER: follower_user
      POSTGRES_PASSWORD: follower_pass
      POSTGRES_DB: follower_db
    volumes:
      - db_follower_1_data:/var/lib/postgresql/data
    depends_on:
      - db_leader
    command: >
      postgres -c hot_standby=on

```

```
  db_follower_2:
    image: postgres:15
    container_name: db_follower_2
    environment:
      POSTGRES_USER: follower_user
      POSTGRES_PASSWORD: follower_pass
      POSTGRES_DB: follower_db
    volumes:
      - db_follower_2_data:/var/lib/postgresql/data
    depends_on:
      - db_leader
    command: >
      postgres -c hot_standby=on

volumes:
  db_leader_data:
  db_follower_1_data:
  db_follower_2_data:
```

Listing 1: Archivo `docker-compose.yml`

Configura Write-Ahead Logging (WAL)

El WAL ya está configurado con los siguientes parámetros en el `db_leader`:

- `wal_level=replica`: Permite replicación.
- `max_wal_senders=5`: Soporta hasta 5 seguidores.
- `wal_keep_size=64MB`: Retiene suficientes registros WAL para sincronizar seguidores.
- `archive_mode=on` y `archive_command='/bin/true'`: Habilita la archivización de logs.

Despliega los servicios

1. Ejecuta Docker Compose:
   - Abre una terminal en la carpeta donde está el archivo `docker-compose.yml`.
   - Ejecuta:

     ```
     docker-compose up -d
     ```

   - Esto descargará las imágenes de PostgreSQL y creará los contenedores.
2. Verifica que los servicios están en ejecución:

   ```
   docker ps
   ```

   Debes ver los contenedores `db_leader`, `db_follower_1` y `db_follower_2`.

Conclusión

Este sistema ahora implementa las siguientes técnicas:

- WAL asegura la integridad de datos.
- Segmentación de logs facilita la administración.
- Leader-Followers asegura redundancia y escalabilidad.

article [utf8]inputenc listings xcolor

Configuración Optimizada de un Proyecto Docker con Python

1. Crear un Directorio para el Proyecto

```
mkdir docker-leader-example
cd docker-leader-example
```

Listing 2: Comando para crear un directorio de trabajo

2. Crear un Archivo `requirements.txt`

```
flask
requests
```

Listing 3: Dependencias del proyecto

3. Crear un Archivo `main.py`

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Docker World!"

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

Listing 4: Código básico en Flask

4. Crear un Dockerfile No Optimizado

4.1 Crear el Archivo Dockerfile

```
nano Dockerfile
```

Listing 5: Abrir el editor para crear el Dockerfile

4.2 Contenido No Optimizado

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y python3 python3-pip
RUN apt-get install -y git
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python3", "main.py"]
```

Listing 6: Dockerfile sin optimizar

4.3 Construir y Ejecutar

```
docker build -t unoptimized-app .
docker run -p 5000:5000 unoptimized-app
```

Listing 7: Construir la imagen y ejecutarla

5. Crear un Dockerfile Optimizado

5.1 Editar el Dockerfile

```
nano Dockerfile
```

Listing 8: Abrir el Dockerfile para aplicar mejoras

5.2 Contenido Optimizado

```
# Usar una imagen base ligera
FROM python:3.9-slim

# Instalar dependencias en una sola capa
RUN apt-get update && apt-get install -y git && \
    rm -rf /var/lib/apt/lists/*

# Copiar primero las dependencias para aprovechar el cache
COPY requirements.txt /app/
WORKDIR /app
RUN pip install -r requirements.txt

# Copiar el c digo fuente restante
COPY . /app

# Comando para ejecutar la aplicaci n
CMD ["python3", "main.py"]
```

Listing 9: Dockerfile optimizado

5.3 Construir y Ejecutar

```
docker build -t optimized-app .
docker run -p 5000:5000 optimized-app
```

Listing 10: Reconstruir y ejecutar el contenedor optimizado

6. Comparar Resultados

Tiempos de Construcción

```
docker build -t unoptimized-app .  # Tiempo: ~60 segundos
docker build -t optimized-app .   # Tiempo: ~30 segundos
```

Listing 11: Comparación de tiempos de construcción

Tamaño de la Imagen

```
docker images
```

Listing 12: Comparación de tamaños de imagen

7. Verificar Funcionamiento

Accede a http://localhost:5000 para comprobar el funcionamiento del servidor en ambas versiones.

8. Conclusión

- La optimización reduce significativamente el tiempo de construcción y el tamaño de la imagen.
- Usar imágenes base ligeras y gestionar correctamente las capas es clave para proyectos Docker eficientes.