Lab 06

Parte 1: Creación del Proyecto y Configuración del `Dockerfile.web`

1. Preparar el Directorio del Proyecto

En CMD, crea un nuevo directorio para tu proyecto y navega a él:

```
mkdir mi-proyecto-docker-compose
cd mi-proyecto-docker-compose
```

2. Crear una Aplicación Web Simple (Node.js)

a. **Inicializar un Proyecto Node.js**:
Crea un nuevo archivo `package.json` :

```
npm init -y
```

b. **Instalar Express** (un marco de servidor web para Node.js):

```
npm install express
```

c. **Crear un Archivo de Aplicación**:
Crea un archivo `index.js` con el siguiente contenido:

```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
  res.send('Hola Mundo desde Docker Compose');
});

app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

Puedes crear este archivo usando vscode

3. Crear el `Dockerfile.web`

a. **Crear un Dockerfile**:

Crea un archivo llamado `Dockerfile.web` en el directorio del proyecto y agrega el siguiente contenido:

```
# Usar la imagen oficial de Node.js como imagen base
FROM node:14

# Establecer el directorio de trabajo en el contenedor
WORKDIR /usr/src/app

# Copiar package.json y package-lock.json (si está disponible)
COPY package*.json ./

# Instalar dependencias del proyecto
RUN npm install

# Copiar los archivos del proyecto al directorio de trabajo del contenedor
COPY . .

# Exponer el puerto en el que la aplicación se ejecutará
EXPOSE 3000

# Comando para ejecutar la aplicación
CMD ["node", "index.js"]
```

Este `Dockerfile` crea una imagen que instala las dependencias de Node.js y ejecuta la aplicación.

4. Crear un Archivo `.dockerignore`

a. **Crear `.dockerignore`** :

Crea un archivo `.dockerignore` para evitar copiar archivos innecesarios al contenedor:

```
node_modules
npm-debug.log
```

Puedes crear este archivo usando un editor de texto.

Parte 2: Configuración de Docker Compose y Ejecución del Proyecto

5. Crear el Archivo `docker-compose.yml`

a. **Crear un archivo `docker-compose.yml`** :

En el directorio del proyecto, crea un archivo `docker-compose.yml` con la siguiente configuración:

```
version: '3.8'
services:
  web:
    build:
      context: .
      dockerfile: Dockerfile.web
    image: mi-web:latest
    container_name: mi-contenedor-web
    ports:
      - "5000:3000"
    volumes:
      - type: bind
        source: ./data
        target: /app/data
    networks:
      mynet:

  db:
    image: postgres
    container_name: mi-contenedor-db
    environment:
      POSTGRES_DB: exampledb
      POSTGRES_USER: exampleuser
      POSTGRES_PASSWORD: examplepass
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      mynet:

networks:
  mynet:

volumes:
  db-data:
```

Explicación de la configuración:

- **web**: Servicio que construye y ejecuta la aplicación Node.js.
- **db**: Servicio de base de datos PostgreSQL.
- **ports**: Mapea el puerto 5000 del host al puerto 3000 del contenedor.
- **networks**: Define una red personalizada `mynet` para la comunicación entre servicios.
- **volumes**: Volumen para la persistencia de datos de PostgreSQL.

6. Construir y Ejecutar con Docker Compose

a. **Construir y Ejecutar**:
En CMD, en el directorio del proyecto, ejecuta:

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

Esto construirá las imágenes si es necesario y lanzará los contenedores.

7. Verificar la Ejecución

a. **Verificar los Contenedores**:
Verifica que los contenedores estén ejecutándose con:

```
docker-compose ps
```

b. **Acceder a la Aplicación Web**:
Abre un navegador y visita `http://localhost:5000` . Deberías ver el mensaje "Hola Mundo desde Docker Compose".

Parte 3: Interacción con la Base de Datos, Volúmenes y Redes

8. Interacción con la Base de Datos

a. **Ejecutar un Cliente de PostgreSQL en Docker**:
Ejecuta un contenedor con cliente de PostgreSQL y conéctate a tu base de datos. Asegúrate de cambiar `exampleuser` , `examplepass` y `mi-contenedor-db` con tus propias credenciales y nombre de contenedor de base de datos.

```
docker run -it --rm --network mi-proyecto-docker-compose_mynet postgres psql -h mi-contenedor-db
```

Una vez dentro del cliente `psql`, puedes ejecutar comandos SQL para interactuar con tu base de datos.

b. **Crear y Consultar Datos**:
Crear una tabla y luego hacer una consulta:

```
CREATE TABLE test(id SERIAL PRIMARY KEY, nombre VARCHAR(50));
INSERT INTO test (nombre) VALUES ('Docker');
SELECT * FROM test;
```

Esto creará una tabla, insertará un registro y luego seleccionará los registros de la tabla.

9. Verificar el Uso de Volúmenes

Para verificar que los datos persisten gracias al uso de volúmenes:

a. **Detener los Servicios**:
Detén los servicios utilizando Docker Compose:

```
docker-compose down
```

b. **Volver a Iniciar los Servicios**:
Vuelve a iniciar los servicios:

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

c. **Verificar Datos**:
Vuelve a conectarte al cliente de PostgreSQL y verifica si los datos aún están presentes:

```
SELECT * FROM test;
```

Deberías ver los datos que insertaste anteriormente, lo que demuestra que los datos persisten en el volumen.

10. Explorar las Redes de Docker

Para explorar cómo los contenedores interactúan en la red `mynet`:

a. **Inspect the Network**:
Inspecciona la red `mynet`:

```
docker network inspect mi-proyecto-docker-compose_mynet
```

Esto te mostrará información sobre la red, incluyendo qué contenedores están conectados a ella.

b. **Prueba de Conectividad entre Contenedores**:
Puedes entrar a uno de tus contenedores y usar herramientas como `ping` o `curl` para probar la conectividad con otros contenedores en la misma red. Por ejemplo:

```
docker exec -it mi-contenedor-web /bin/sh
ping mi-contenedor-db
```

Esto demuestra que los contenedores pueden comunicarse entre sí utilizando los nombres de los contenedores como nombres de host dentro de la red de Docker.