Implementar pila LAMP con docker-compose

¿Qué es y para qué sirve docker-compose?

"docker-compose" es una herramienta de Docker que permite definir y ejecutar aplicaciones **multicontenedor**. Con docker-compose, puedes gestionar servicios complejos (como una pila LAMP) desde un único archivo de configuración, lo que **facilita la creación, configuración y orquestación de múltiples contenedores relacionados en una sola red de Docker**.

Principales Características de docker-compose

- **Simplificación de la Configuración**: docker-compose utiliza un archivo de configuración (`docker-compose.yml`) donde defines todos los servicios que forman parte de la aplicación, incluyendo sus redes, volúmenes y variables de entorno.
- **Orquestación Multicontenedor**: Permite gestionar múltiples contenedores con un solo comando, evitando la necesidad de iniciar y configurar cada contenedor de forma individual.
- **Persistencia de Datos**: Permite especificar volúmenes que mantienen los datos entre reinicios de contenedores o versiones.
- **Escalabilidad**: docker-compose facilita la creación de múltiples instancias de un servicio para gestionar cargas, en casos en los que se necesite escalado horizontal.

¿Cómo Funciona docker-compose?

docker-compose se basa en un archivo de configuración **YAML** (`docker-compose.yml`) que especifica los servicios (contenedores), redes y volúmenes necesarios para ejecutar una aplicación completa. Al ejecutar el comando `docker-compose up`, la herramienta interpreta el archivo y despliega los contenedores definidos, configurando las redes y montando los volúmenes según las especificaciones.

Estructura de un Archivo `docker-compose.yml`

El archivo `docker-compose.yml` tiene una estructura definida para que puedas especificar todos los aspectos de cada servicio:

- **version**: Define la versión de docker-compose que se está usando (por ejemplo, `version: "3.8"`).
- **services**: Define los contenedores que formarán parte de la aplicación. Cada servicio puede configurarse con:
  - **image**: La imagen de Docker que se usará.
  - **build**: Instrucciones para construir una imagen a partir de un `Dockerfile`.
  - **ports**: Los puertos expuestos en el contenedor y su redirección al anfitrión.
  - **volumes**: Montajes de volumen para persistir datos o compartir archivos entre el sistema anfitrión y el contenedor.
  - **networks**: Redes a las que se conectará el contenedor.
  - **environment**: Variables de entorno para configurar el contenedor.

- **volumes**: Define volúmenes compartidos o persistentes para almacenar datos entre reinicios de los contenedores.
- **networks**: Define las redes a las que pertenecen los contenedores, facilitando la comunicación entre ellos.

Instalación de la herramienta

docker-compose no viene incluido automáticamente con la instalación del cliente Docker en todas las plataformas, aunque **en versiones más recientes de Docker Desktop (para Windows y macOS), docker-compose se incluye por defecto**. Sin embargo, en sistemas basados **en Linux**, generalmente **necesitas instalarlo de forma independiente**.

Aquí tienes detalles sobre su disponibilidad según el sistema operativo:

1. Docker Desktop (Windows y macOS)

Si estás usando Docker Desktop en Windows o macOS, docker-compose ya viene instalado y está disponible para usarse directamente desde la terminal.

- Verifica la versión ejecutando:

```
docker-compose --version
```

2. Linux

En Linux, necesitas instalar docker-compose de forma independiente, ya que no se incluye por defecto en la instalación de Docker Engine.

Para instalar docker-compose en Linux, puedes seguir estos pasos:

1. Descargar la versión más reciente desde GitHub:

```
sudo curl -L "https://github.com/docker/compose/releases/download/$(curl -s https://api.github.com/repos/docker/compose/releases/latest | grep tag_name | cut -d '"' -f 4)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Hacer que docker-compose sea ejecutable:

```
sudo chmod +x /usr/local/bin/docker-compose
```

3. Verificar la instalación:

```
docker-compose --version
```

3. Docker Compose V2

Desde Docker Compose V2, la funcionalidad de docker-compose se ha integrado en el propio comando `docker` como un subcomando (`docker compose`). En esta versión, puedes usar `docker compose` en lugar de docker-compose.

Para los usuarios de Docker Desktop, `docker compose` está habilitado automáticamente. En Linux, si tienes Docker Engine actualizado a la última versión, deberías poder usar el nuevo comando `docker compose` sin instalaciones adicionales.

Ejemplo Básico de un Archivo `docker-compose.yml`

Imaginemos que queremos configurar una pila básica con una aplicación web que se conecta a una base de datos MySQL. A continuación, se muestra un archivo `docker-compose.yml` para esta configuración:

```yaml
version: "3.8"

services:
  app:
    image: php:8.2-apache
    volumes:
      - ./html:/var/www/html
    ports:
      - "8080:80"
    depends_on:
      - db

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root-password
      MYSQL_DATABASE: mydb
      MYSQL_USER: user
      MYSQL_PASSWORD: user-password
    volumes:
      - db_data:/var/lib/mysql

volumes:
  db_data:
```

En este ejemplo:
- **app**: Es el servicio que ejecuta una aplicación PHP con Apache, montando la carpeta `html` del sistema anfitrión en el directorio `/var/www/html` del contenedor.

- **db**: Es el servicio que ejecuta MySQL con una configuración específica de base de datos y credenciales.
- **volumes**: Define `db_data`, un volumen persistente para almacenar los datos de MySQL y evitar que se pierdan entre reinicios.

Comandos Básicos de docker-compose

1. `docker-compose up`

Inicia todos los servicios definidos en `docker-compose.yml`. Si usas el flag `-d`, los servicios se ejecutarán en segundo plano.

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

2. `docker-compose down`

Detiene y elimina los contenedores, redes y volúmenes definidos en el archivo. No afecta los volúmenes de datos a menos que se use el flag `-v`.

```
docker-compose down
```

3. `docker-compose build`

Construye las imágenes a partir de los `Dockerfile` especificados en el archivo `docker-compose.yml`.

```
docker-compose build
```

4. `docker-compose logs`

Muestra los logs de todos los servicios definidos. Puedes ver los logs de un servicio específico al agregar el nombre del servicio después del comando.

```
docker-compose logs app
```

5. `docker-compose exec`

Ejecuta comandos en un contenedor en ejecución. Por ejemplo, para abrir una shell en el contenedor `app`:

```
docker-compose exec app bash
```

Ventajas de Usar docker-compose

- **Automatización de Entornos**: Permite automatizar la configuración de entornos de desarrollo o producción de manera rápida y reproducible.
- **Consistencia en la Configuración**: La configuración de todos los servicios se define en un solo archivo, lo que facilita la revisión y actualización de la infraestructura.
- **Compatibilidad Multicontenedor**: Ideal para aplicaciones que requieren varios contenedores que interactúan entre sí, como una aplicación web con bases de datos, caché, balanceadores, etc.

Veamos ahora un tutorial completo para implementar una pila LAMP usando docker-compose, en la que se configuran dos sitios web con sus respectivos virtual hosts en Apache y se incluye un contenedor MySQL para almacenar datos.

Paso 1: Estructura de Archivos del Proyecto

Primero, organiza tu proyecto en el sistema de archivos. La estructura será similar a la siguiente:

```
project-directory/
│
├── docker-compose.yml
├── apache/
│   └── sites-available/
│       ├── sitio1.com.conf
│       └── sitio2.com.conf
├── html/
│   ├── sitio1/
│   │   └── public/
│   │       └── index.html
│   └── sitio2/
│       └── public/
│           └── index.php
└── mysql/
    └── init.sql
```

Paso 2: Crear el Archivo `docker-compose.yml`

Este archivo define y configura los servicios de la pila LAMP: Apache con PHP, MySQL, y phpMyAdmin. Cada servicio se ejecutará en un contenedor separado dentro de una red Docker.

```yaml
version: '3.8'

services:
  webserver:
    image: php:8.2-apache
    container_name: webserver
    volumes:
      - ./html:/var/www/html
      - ./apache/sites-available:/etc/apache2/sites-available
    ports:
      - "8080:80"
    networks:
      lamp-network:
        ipv4_address: 192.168.200.100
```

```
    depends_on:
      - db
    environment:
      - MYSQL_HOST=db
      - MYSQL_DATABASE=mydb
      - MYSQL_USER=user
      - MYSQL_PASSWORD=user-password
    command: >
      /bin/bash -c "
      ln -s /etc/apache2/sites-available/sitio1.com.conf
/etc/apache2/sites-enabled/ &&
      ln -s /etc/apache2/sites-available/sitio2.com.conf
/etc/apache2/sites-enabled/ &&
      chown -R www-data:www-data /var/www/html &&
      chmod -R 755 /var/www/html &&
      apache2-foreground
      "

  db:
    image: mysql:8.0
    container_name: mysql-container
    volumes:
      - ./mysql:/docker-entrypoint-initdb.d
    environment:
      MYSQL_ROOT_PASSWORD: root-password
      MYSQL_DATABASE: mydb
      MYSQL_USER: user
      MYSQL_PASSWORD: user-password
    networks:
      lamp-network:
        ipv4_address: 192.168.200.200

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: root-password
    ports:
      - "8081:80"
    depends_on:
      - db
    networks:
```

```
      lamp-network:
        ipv4_address: 192.168.200.201

networks:
  lamp-network:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.200.0/24
```

Desglose del fichero yaml:

- `**webserver**`: Contenedor que ejecuta Apache con PHP. Monta los directorios `html` y `sites-available` para los archivos de los sitios y la configuración de Apache.
- `**command**`: Ejecuta enlaces simbólicos para habilitar los sitios, asigna permisos y luego inicia Apache en primer plano.
- `**db**`: Contenedor de MySQL que crea la base de datos `mydb` con el usuario `user`.
- `**phpmyadmin**`: Contenedor para administrar la base de datos MySQL, accesible en el puerto 8081.

Paso 3: Configuración de los Virtual Hosts para Apache

Crea los archivos de configuración de Apache en `**apache/sites-available/**` para los dos sitios.

`sitio1.com.conf`:

```
<VirtualHost *:80>
    ServerName sitio1.com
    ServerAlias www.sitio2.com
    DocumentRoot /var/www/html/sitio1/public

    <Directory /var/www/html/sitio1/public>
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

`sitio2.com.conf`:

```
<VirtualHost *:80>
    ServerName sitio2.com
    ServerAlias www.sitio2.com
    DocumentRoot /var/www/html/sitio2/public

    <Directory /var/www/html/sitio2/public>
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Paso 4: Archivos de los Sitios Web

En la carpeta `**html/**` debes colocar los archivos de cada sitio:

- `**html/sitio1/public/index.html**`:

```
<!DOCTYPE html>
  <html lang="es">
  <head>
      <meta charset="UTF-8">
      <title>Sitio 1</title>
  </head>
  <body>
      <h1>Bienvenido a Sitio 1</h1>
      <p>Este es un sitio web estático.</p>
  </body>
  </html>
```

- `**html/sitio2/public/index.php**`:

```php
<?php
  echo "<h1>Bienvenido a Sitio 2</h1>";
  echo "<p>Este es un sitio web dinámico.</p>";

  $servername = getenv('DB_HOST');
  $username = getenv('DB_USER');
  $password = getenv('DB_PASS');
  $dbname = getenv('DB_NAME');

  $conn = new mysqli($servername, $username, $password, $dbname);

  if ($conn->connect_error) {
      die("Conexión fallida: " . $conn->connect_error);
  }

  echo "<p>Conexión a la base de datos exitosa</p>";
  $conn->close();
  ?>
```

Paso 5: Inicializar la Base de Datos (Archivo `init.sql` en `mysql/`)

Este archivo contiene la configuración inicial para MySQL. Si deseas crear una tabla de ejemplo, incluye una configuración básica:

```sql
CREATE TABLE IF NOT EXISTS test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);

INSERT INTO test (name) VALUES ('PHP'), ('MySQL'), ('Docker');
```

Paso 6: Modificar el Archivo `hosts` (opcional para Pruebas Locales)

Para probar los sitios `sitio1.com` y `sitio2.com` localmente, agrega estas líneas en el archivo `**hosts**` de tu sistema anfitrión:

```
127.0.0.1 sitio1.com
127.0.0.1 sitio2.com
```

Esto permitirá que los nombres de dominio `sitio1.com` y `sitio2.com` apunten a `localhost`.

Paso 7: Ejecutar docker-compose

Ejecuta el siguiente comando en el directorio principal para construir y ejecutar la pila LAMP:

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

Paso 8: Acceder a los Sitios y phpMyAdmin

- Sitio 1: `**http://sitio1.com:8080**`
- Sitio 2: `**http://sitio2.com:8080**`
- phpMyAdmin: `**http://localhost:8081**` (usuario: `root`, contraseña: `root-password`)

En Resumen

Este tutorial configura una pila LAMP con dos sitios web y un contenedor MySQL administrado con phpMyAdmin, todo orquestado con **docker-compose**. La configuración de los hosts virtuales de Apache permite servir cada sitio web en un dominio diferente dentro del entorno de Docker.