SPOTIFY TO VYNIL

HECHO POR:

- ELÍAS GONZÁLEZ VALDEPEÑAS (elias.gonzalez@alu.uclm.es)
- RODRIGO DE LA HOZ LÓPEZ (rodrigo.de@alu.uclm.es)
- ALBERTO PALENCIA QUIÑÓNES (alberto.palencia1@alu.uclm.es)

Desarrollador Backend -> Alberto Palencia

Desarrollador Frontend -> Elías González

Desarrollador Testing -> Rodrigo de la Hoz

Enlace GitHub

https://github.com/AlbertoPalenciaQuinones/ISI-LAB.git

Contenido

1. Objetivo del Sprint 4

El objetivo principal de este sprint es preparar y configurar el proyecto para su despliegue en un entorno cloud, garantizando su portabilidad, escalabilidad y facilidad de mantenimiento. Para ello, se ha realizado la contenerización completa del servicio web mediante Docker y Docker Compose, separando la aplicación Flask del servicio de base de datos MySQL.

Además, se ha definido un entorno de hosting en la nube compatible con imágenes Docker, permitiendo que el proyecto pueda ser desplegado de forma automática en cuestión de minutos. El enfoque se ha centrado en facilitar un despliegue rápido y reproducible, tanto localmente como en plataformas cloud, asegurando que todos los componentes del sistema funcionen correctamente de forma integrada.

2. Tecnologías utilizadas

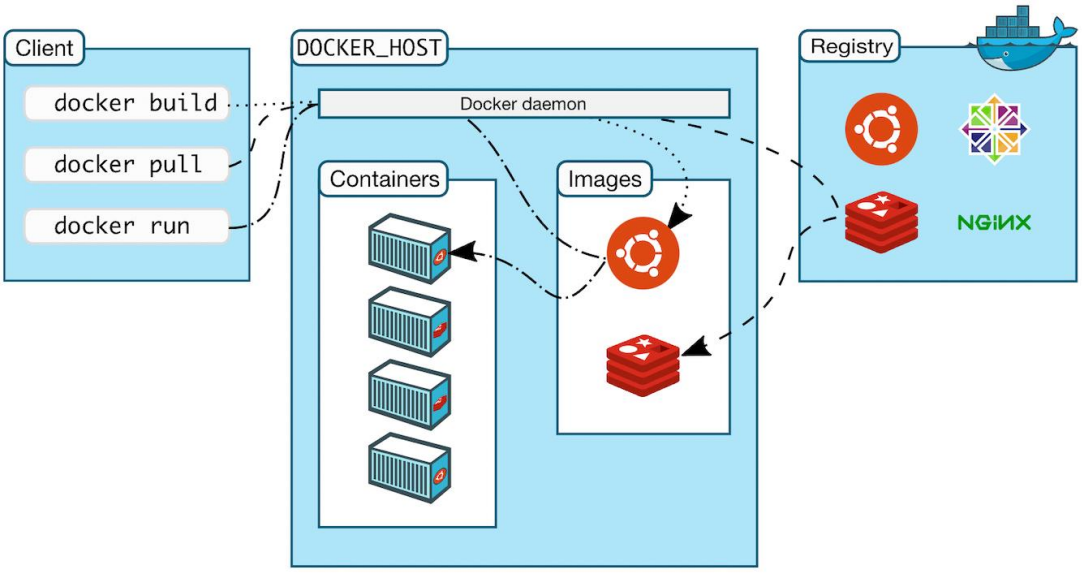
- **Backend**:
  - Python, Flask
- **Base de datos**:
  - MySQL 8.0
- **APIs externas**:
  - Discogs, LastFM
- **Frontend**:
  - HTML
- **Contenerización**:
  - Docker
- **Sistema operativo base**:
  - Debian

3. Docker

3.1. Descripción

La contenerización consiste en encapsular la aplicación y sus dependencias en contenedores Docker, permitiendo que se ejecute de forma consistente en cualquier entorno. En este sprint se ha creado una arquitectura de contenedores para:

- Garantizar el despliegue rápido y reproducible
- Separar responsabilidades entre la aplicación web y la base de datos
- Facilitar la escalabilidad y el mantenimiento



3.2. Estructura de contenedores

Se ve el número de contenedores utilizados y el porque de cada uno. En nuestro proyecto, hemos utilizado un total de 2 contenedores:

- Contenedor web (**web**): Su función es ejecutar la aplicación flask (app.py), las dependencias se instalan a través de requirements.txt. Se iniciará en el puerto “puerto” y se conecta con el contenedor de la base de datos.

- Contenedor base de datos (db): Su función es ejecutar un servidor MySQL 8.0. Su configuración se hace mediante el nombre de usuario de la base de datos + contraseña, situados en el archivo config.py. El nombre de la base de datos es musicfinder

3.3. Archivos clave

Los archivos utilizados para realizar la contenerización son:

- **Dockerfile**: define como construir el contenedor web, es una secuencia de instrucciones.
- **docker-compose.yml**: información general del contenedor, define redes, puertos, dependencias…
- **requirements.txt**: lista las dependes de Python necesarias.
- **config.py**: gestiona aspectos de la configuración Flask y conexión a MySQL como el usuario, contraseña, host, puertos…

3.4. Ventajas de usar Docker

El utilizar contenedores en nuestro proyecto presenta una serie de ventajas para el mismo:

- **Separación de servicios**: cada contenedor cumple una función específica.
- **Compatibilidad**: no importa del sistema operativo en el que estemos, la aplicación se va a poder ejecutar igual gracias al Docker.
- **Configuración mínima**: al empaquetar la aplicación entera en un Docker, para desplegarla, solo es necesario tener instalado Docker y Compose.
- **Persistencia de datos**: la base de datos utiliza volúmenes para conservar información, aunque el contenedor se detenga.

4. Hosting en la nube

Para desplegar el proyecto en un entorno accesible desde Internet, se ha optado por utilizar un servicio de hosting en la nube que permita ejecutar contenedores Docker y brindar una infraestructura estable, escalable y de fácil administración. La solución elegida ha sido **Hostinger**, un proveedor reconocido por su rendimiento, soporte técnico y relación calidad-precio.

Dado que el proyecto está contenerizado mediante Docker, era imprescindible contar con un entorno que ofreciera acceso root, soporte para instalación de Docker, y la posibilidad de gestionar puertos, volúmenes y redes a medida. Por esta razón, se ha seleccionado un servidor VPS que cumple con estos requisitos, garantizando un despliegue completo, reproducible y accesible desde cualquier dispositivo conectado a Internet.

4.1. Hostinger

Hostinger es un proveedor de servicios de hosting web reconocido por su facilidad de uso, tarifas asequibles y su soporte para tecnologías modernas como Docker, VPS y contenedores personalizados. Es una buena elección para desplegar aplicaciones web de backend como la tuya, basada en Flask + MySQL.

4.2. Tarifa seleccionada

Para el despliegue del proyecto, se ha elegido el plan **Cloud Startup** de Hostinger, dentro de su gama de Cloud Hosting gestionado. Este tipo de servicio proporciona una infraestructura más potente y escalable que el hosting compartido, sin llegar a la complejidad de un VPS autogestionado.

Decidimos contratar la tarifa para 12 meses, creemos que es el tiempo suficiente para poder comprobar si la aplicación va a dar impacto en el mercado y si se le puede sacar partido.

Las tarifas de Hostinger quedan en el siguiente enlace: https://www.hostinger.com/es/precios

Detalles del plan Cloud Startup:

Característica	**Valor**
Plan	Cloud Startup
Precio	9.99€/mes
vCPU	3 núcleos
RAM	3GB
Almacenamiento SSD	200GB
Ancho de banda	Ilimitado
¿Dominio gratuito?	Sí
¿IP dedicada?	Sí
Soporte Docker	Parcial (SSH y Git)
Panel de control	hPanel (propietario de hostinger)

4.3. Justificación de la tarifa

Hay una serie de características por las que hemos decidido seleccionarla tarifa Cloud Startup:

- Más potencia para entornos reales: al contener varios servicios la aplicación (Flask y MySQL), la tarifa beneficia la misma debido a su mayor RAM y CPU frente al hosting compartido.

- Alta disponibilidad y rendimiento: Cloud Hosting está basado en la infraestructura cloud, lo que garantiza carga más rápida y mejor tiempo de actividad. Esto es algo fundamental para las aplicaciones con bases de datos.

- Escalabilidad: permite escalar fácilmente a planes superiores, si las características de Cloud Startup no son suficientes en un futuro, se podrá cambiar a un plan superior sin que suponga ningún problema.

5. Plan de despliegue

Se detalla un plan de despliegue para mostrar la incorporación de la aplicación en cualquier sistema.

Requisitos:

- Instalar Docker Desktop y crear una cuenta para usarlo.

1. Abre una terminal y navega hasta la carpeta 'src' del proyecto.
2. Ejecuta el siguiente comando para construir y levantar los contenedores: docker-compose up –build mientras que tienes abierto Docker Desktop.
3. Espera a que los servicios estén completamente levantados.
4. Accede a la aplicación desde tu navegador en http://localhost:5000.

Una vez desplegada, la aplicación está disponible en http://localhost:5000, donde se puede interactuar con las funcionalidades de búsqueda, gestión de usuarios, álbumes y wishlist.

Para detener y eliminar los contenedores y los volúmenes asociados, ejecuta: docker-compose down -v