Entrega 1 – Frontend + Backedn Rest + Procesamiento Asíncrono Aplicaciones Web Escalables en un entorno tradicional

Paul A. Calvache Tapia, Fabian O. Ramírez, David A. Vásquez Pachón, Mateo Zapata Lopez
Arquitectura, conclusiones y consideraciones
Universidad de los Andes, Bogotá, Colombia
{pa.calvache, fo.ramirez50, da.vasquez11, m.zapatal2}@uniandes.edu.co
Fecha de presentación: febrero 24 de 2023

Tabla de contenido

1 Introducción

La empresa “cloud conversión tool” desea crear una aplicación que será ofrecida gratuitamente en internet con varios requerimientos. En esta solución tiene un aplicativo para comprimir archivos en los formatos ZIP, GZ y BZ2. Esta aplicación se adhiere a los requerimientos de la compañía solicitante.

El lenguaje de programación Python es interpretativo y posee librerías que permiten desarrollar aplicaciones web con diferentes funcionalidades. Con este lenguaje se desarrolla un APIREST la cual se conecta a una base de datos, y se encuentra en un contenedor de Docker para su funcionamiento. En esta entrega se desarrolló una aplicación para para 8 Endpoints (se define como acción específica sobre la aplicación), utilizando los métodos GET, POST, PUT y DELETE.

Descripción de los Endpoints:

1. /api/auth/signup (POST)
   Permite crear una cuenta de usuario, con los campos usuario, correo electrónico y 2 campos para contraseña (ingreso y confirmación). El usuario y el correo electrónico deben ser únicos en la plataforma y siguen un proceso de requerimientos básicos de nulos, para todos los datos ingresados un proceso de HASH a toda la información para guardarlo en la base de datos.
   Cuando se registra un usuario nuevo se crean 2 carpetas, una en la carpeta archivos y otra en la carpeta archivosComprimidos.

2. /api/auth/login (POST)
   Permite recuperar el token de autorización para consumir los recursos del API suministrando un nombre de usuario y una contraseña correcta de una cuenta registrada, y se redirecciona a el Endpoint Cargar Archivo.

3. /api/tasks (GET)
   Permite recuperar la lista todas las tareas de conversión de un usuario autorizado en la aplicación.

4. /api/tasks (POST)
   Permite crear una nueva tarea de conversión de formatos subiendo un archivo y especificando a qué formato se quiere comprimir. El usuario requiere autorización bearer token.

5. /api/tasks/<int:id_task> (GET)
   Permite recuperar la información de una tarea en la aplicación. El usuario requiere autorización bearer token.

6. /api/tasks/<int:id_task> (DELETE)
   Permite eliminar una tarea en la aplicación. El usuario requiere autorización bearer token.

7. /api/files/<filename> (GET)
   Permite recuperar el archivo original o procesado.

8. /api/files (POST)
   Permite cargar archivos a una ruta especifica.

9. /api/files/delete/<string:filename> (DELETE)
   Permite eliminar archivos de una ruta especifica.

10. /api/files/comprimir (POST)
    Permite comprimir archivos de 3 maneras diferentes y enviar correos de confirmación.

2 Arquitectura de la aplicación

La arquitectura de la aplicación se muestra en la imagen 1, y a continuación se describen los detalles de su arquitectura:

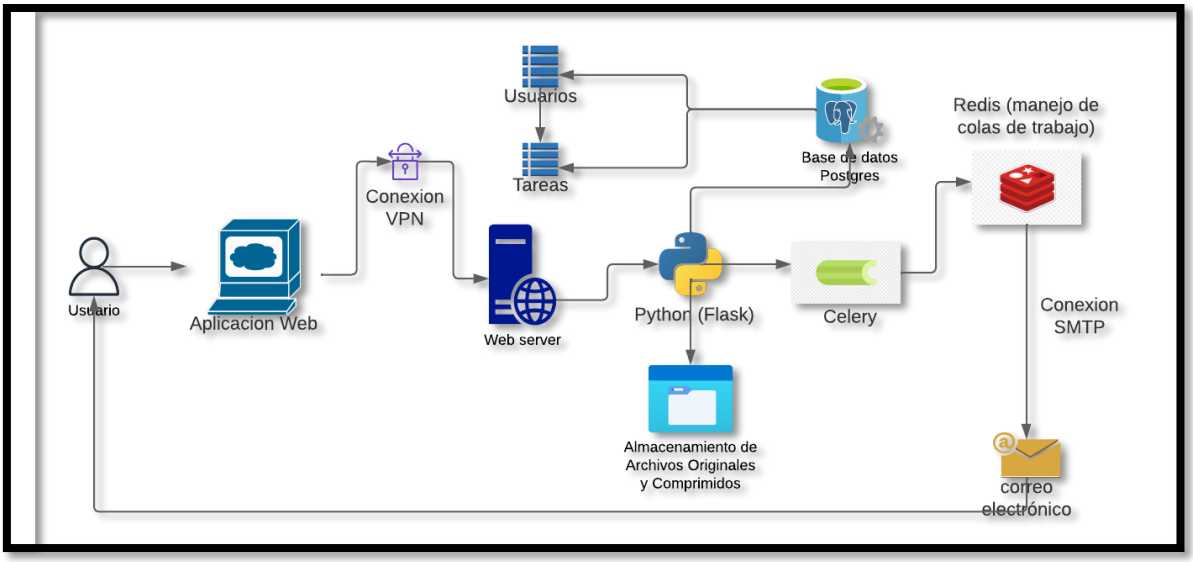

Imagen 1. Arquitectura aplicación web tradicional.

La aplicación web se desarrolló utilizando el framework Flask, el cual se provee tanto el Frontend de la aplicación a través de modelos HTML y CSS como su Backend, representado en la lógica de negocio, gestión de base de datos a través del ORM (Object-Relational Mapping) SQL Alchemy y la gestión de tareas en segundo plano a través de Celery.

El usuario se conecta a través del puerto 5000 para conectarse a la aplicación web. En la cual podrá registrar un usuario con email y nombre de usuario único, luego será dirigido a la página de login para ingresar a su cuenta (se crea un token de autenticación invisible para el usuario), posteriormente el usuario podrá navegar y cargar archivos, crear tareas, listar sus tareas, eliminar archivos y comprimir archivos.

Para la creación de usuarios, tareas y listar tareas se realiza mediante SQL Alchemy, realizando las acciones en la base de datos Postgres. La creación de tareas se realiza después de haber cargado archivos que deseen comprimir y tendrán estado “uploaded” para ser procesados en Redis (manejo de colas de trabajo, procesamiento asíncrono) el cual realizara la acción de comprimir en diferentes formatos como ZIP, GZ y BZ2, y finalmente enviar una notificación al usuario confirmando que el archivo está disponible para su descarga.

3 Infraestructura utilizada

La infraestructura utilizada para la ejecución de la aplicación es una máquina virtual que se obtiene a través de AWS (Amazon Web Services). La máquina virtual ejecuta el sistema operativo Ubuntu 20.04 y dispone de un almacenamiento de 32 GBs aproximadamente, memoria RAM de 2 GBs, Swap de 4 GBs y procesador Intel Xeon, cuyos procesadores corren a 2.60GHz.

4 Limitaciones de escalabilidad y desarrollo

Las limitaciones por escalabilidad y desarrollo, flask permite crear APIs escalables, sin embargo, en este caso si existen limitaciones de escalabilidad ligadas a la infraestructura de despliegue, para ser puntuales, la capacidad de las máquinas virtuales donde se ejecuta. Debido a lo anterior, se podría obtener una escalabilidad vertical migrando todo el sistema a un nuevo hardware más potente y eficaz que con el que se cuenta. Sin embargo, una escalabilidad horizontal no se podría lograr en este escenario, pues se deberían añadir más equipos para ampliar su potencia de trabajo.

En cuanto al manejo de colas la compresión de archivos pequeños como imágenes, archivos de texto, entre otros se realiza en aproximadamente 10 segundos o menos, lo cual permite notificar al usuario en un corto periodo de tiempo.

5 Conclusiones

- La librería y marco de trabajo de flask es muy flexible y escalable para crear diferentes Endpoints de una manera ordenada y entendible, Docker nos permite realizar despliegues en la nube, almacenando nuestras aplicaciones en contenedores y facilitar obtener los servicios como Redis, Postgres y muchos más.
- Visualizar los Endpoints en POSTMAN permite ver la funcionalidad de nuestro código y facilitar la creación del Backend y Frontend, observando el comportamiento total de nuestro código.
- Separar la creación de tareas de subir archivos es un punto positivo para la arquitectura deseada, ya que facilita que el usuario no ingrese información errónea al sistema de nombres de archivos,

la separación de archivos en carpetas únicas, la creación de un token para realizar acciones en la aplicación adiciona seguridad para los usuarios de solo obtener información de su actividad.

- Otro aspecto importante a resaltar es el óptimo uso de la librería de Bootstrap con Flask, esta integración, permite el ágil desarrollo de aplicaciones web de forma rápida y completa, pues la curva de aprendizaje es rápida.

6 Bibliografía

- The New Flask Mega-Tutorial, Miguel Grinberg, https://blog.miguelgrinberg.com/index, [Recurso Web].
- Tutorial - Docker, Jesse Padilla, https://docs.docker.com/, [Recurso en línea].
- Redis, https://hub.docker.com/_/redis, [Recurso en línea].
- Celery Documentación, https://docs.celeryq.dev/en/stable/getting-started/introduction.html, [Recurso en línea].