Documentación Desafío Técnico

1. Descripción

Es un microservicio que expone un endpoint para leer y procesar un archivo, consultar una serie de APIs públicas de MercadoLibre y cargar los registros a una base de datos.

2. Características técnicas

2.1 Datos básicos

- La aplicación está basada en el framework Flask de Python y utiliza una base de datos con el motor Postgres.
- El microservicio incorpora un orquestador de servicios que ejecuta cada etapa del proceso como un componente que se implementa como servicio.
- El endpoint del servicio de carga de archivos (loader) implementa un validador para controlar los parámetros enviados en la request.
- La ejecución del orquestador es controlada por medio de reglas en formato Json.
- El proyecto de desarrollo de la aplicación fue alojado en el repositorio GitHub - pipemohz/loader-microservice de Github.

2.2 Arquitectura de la aplicación

La estructura de la aplicación está diseñada para agregar procesos como aplicaciones del proyecto. En cada proceso se implementan etapas como componentes que incorporan un servicio basado en un servicio base abstracto.

Los endpoint del microservicio son configurados en un paquete denominado *api,* en donde se definen las implementaciones de las rutas y las respuestas que se entregan a las solicitudes de cada endpoint.

La configuración de la aplicación se lleva a cabo en el paquete denominado *config*. En este se define un archivo central para la lectura de variables de entorno y un archivo de parámetros de la aplicación Flask.

A continuación se presenta un diagrama de la Function App

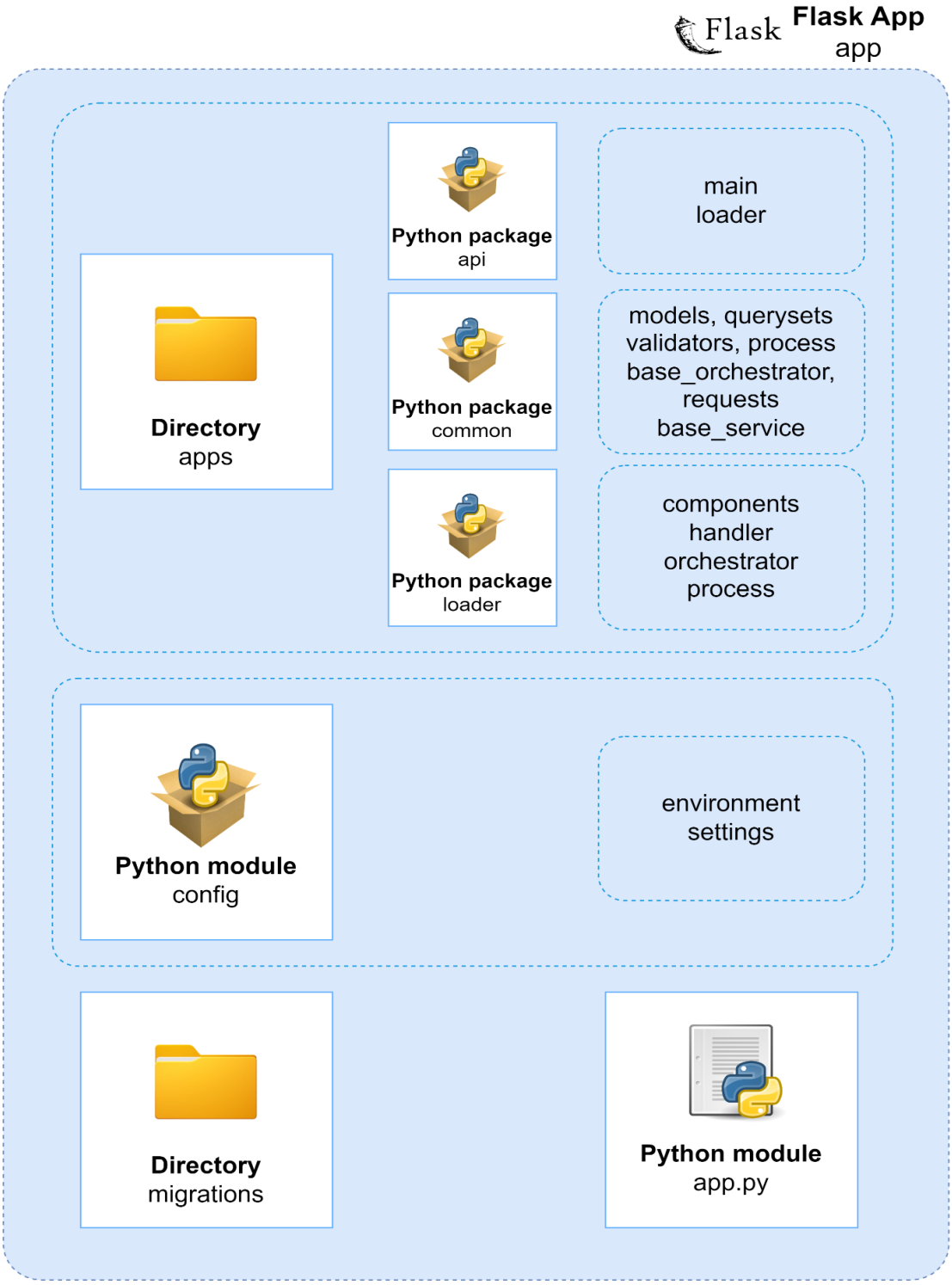

Figura 1: Diagrama de aplicación

3. Procesos de la aplicación

La aplicación realiza un proceso de carga y lectura de un archivo y ejecución de requests HTTP a un grupo de APIs de mercado libre. El proceso está consolidado en el paquete denominado loader. Este está dividido en tres componentes: Un formateador del archivo (formatter), un lector de archivos (reader) y un cliente para lanzar las requests (client).

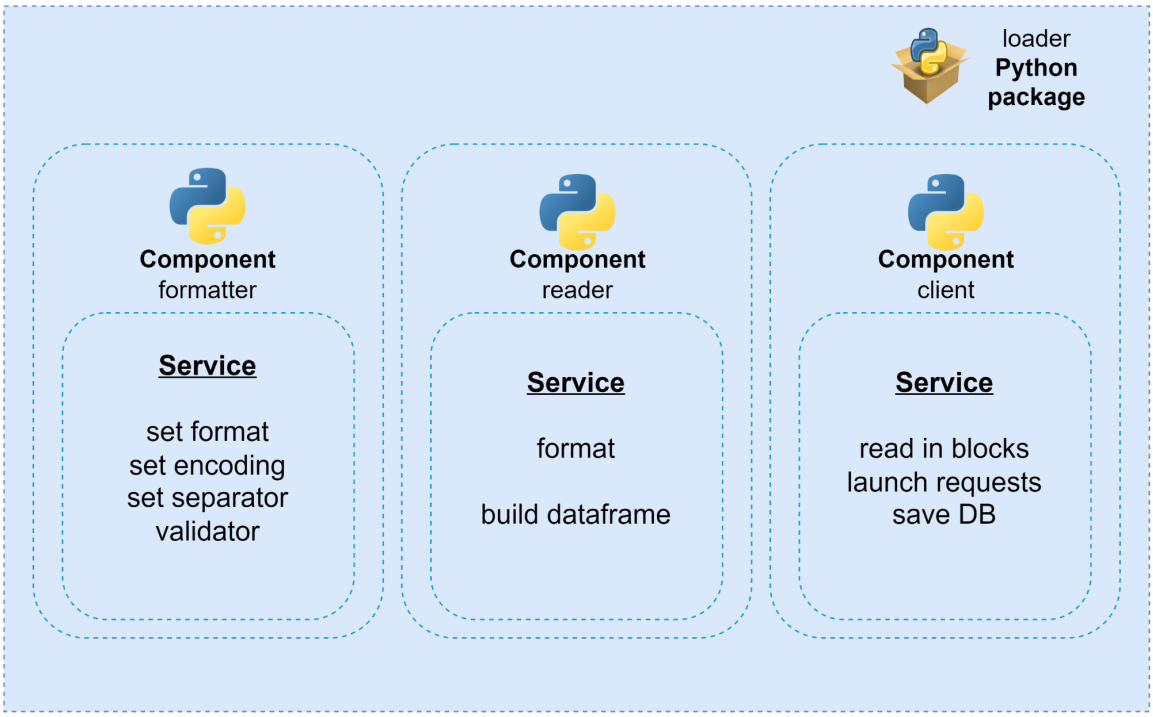

Figura 2. Procesos de la aplicación.

3.1 Componente Formatter:

Este servicio está encargado de configurar las propiedades del archivo leído y hacer validación de los parámetros especificados por el usuario al consumir el endpoint de ejecución del proceso bajo demanda.

El validador está basado en una clase que hereda del modelo base de la librería pydantic. Las reglas están definidas para ser controladas por medio de variables de entorno del proyecto.

3.2 Componente Reader:

La función de este servicio es procesar el stream de datos del archivo a partir de las propiedades definidas por el componente formatter. Entrega un dataframe de pandas como salida.

En esta etapa se verifican tipos de datos en el archivo, estructura de los datos, y la integridad de los registros. Se eliminan aquellos que no están formateados correctamente, o que contengan datos nulos.

3.3 Componente Client:

Este componente tiene como función generar las requests HTTP por cada registro del dataframe entregado por el componente reader.

Las requests se generan en una piscina de hilos para realizar las cuatro tareas requeridas por cada registro del dataframe.

El archivo se procesa en bloques de 20 registros cuyos datos se incluyen en una request a la API de ítems de mercado libre. A partir de la respuesta se disparan 3 requests concurrentes por cada registro a las APIs de currencies, categories y users. Una vez se obtienen las respuestas y se actualizan los valores en los registros se escriben en la base de datos.

4. Despliegue de la aplicación en local

Requisitos previos para realizar el despliegue del proyecto:

- Python 3.10.x
- Motor Docker
- Contenedor de docker postgres
- Crear una base de datos en postgres para el proyecto

A continuación se describen los pasos para despegar la aplicación en local para realizar pruebas:

- Descargar el proyecto desde el repositorio alojado en Git hub
- Crear un entorno virtual de python en la carpeta raíz del proyecto
- Activar el entorno virtual
- Instalar las dependencias requeridas especificadas dentro del archivo requirements.txt
- Configurar un archivo de configuración de las variables de entorno (.env). Use como referencia el archivo .env.example

- Ejecutar los test sobre los componentes del servicio loader. Corra el script run_tests.sh
- Si los test se ejecutan con éxito la configuración del microservicio está completa.
- Desplegar en local el microservicio con el comando python app.py

5. Endpoints del microservicio

El microservicio tiene 3 endpoints: index, health, on_demand.

- El index responde con status 200 y el nombre del microservicio en el mensaje.
- health responde con status 200 y el mensaje "OK"
- on_demand tiene cuatro tipos de respuesta:

  1. Respuesta exitosa código HTTP 200 y entrega el mensaje “success”

  2. Respuestas de error código HTTP 500 entrega un mensaje con el error de ejecución en el servicio.

  3. Respuesta error de formato código HTTP 400 entrega un mensaje especificando un error en los parámetros de la solicitud (format, separator, encoding). Verifica que no se ingresen campos extra en la solicitud.

  4. Respuesta de solicitud incorrecta código HTTP 400 entrega un mensaje especificando errores en el procesamiento del archivo (contenido del archivo y estructura de los datos).

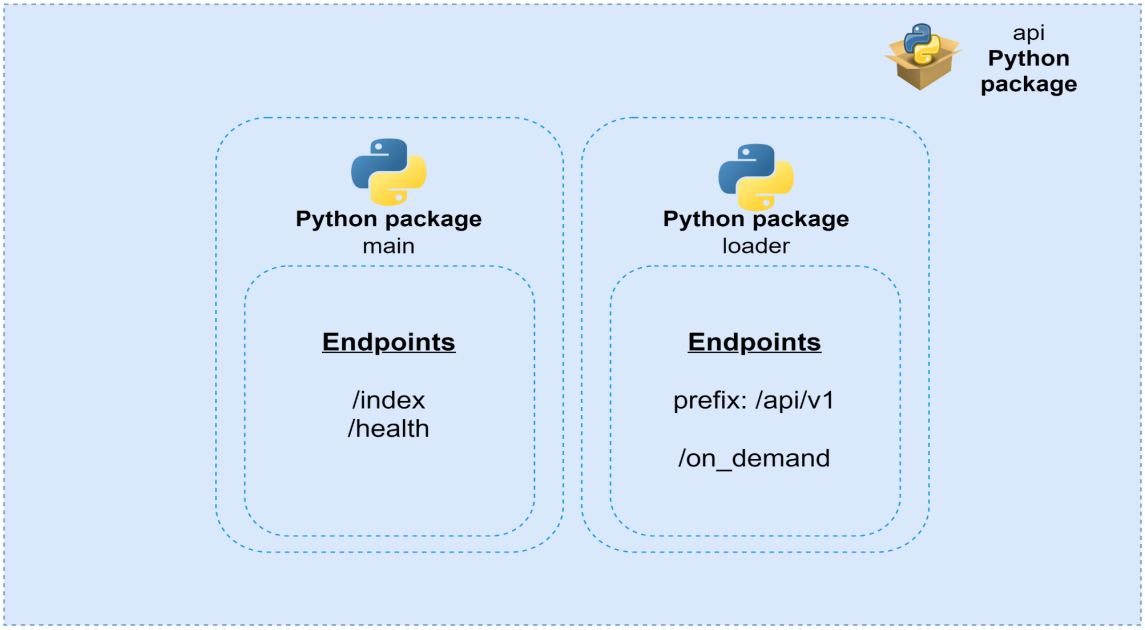

Figura 3: Endpoint del microservicio

Desafío Teórico

Consigna

Procesos, hilos y corrutinas

● Un caso en el que usarías procesos para resolver un problema y por qué.
R/ La programación de procesos es la actividad del administrador de procesos que maneja la eliminación del proceso en ejecución de la CPU y la selección de otro proceso sobre la base de una estrategia particular. Los procesos son una parte esencial de los sistemas operativos de multiprogramación.

Caso: Se puede utilizar para hacer codificación y decodificación de información por ejemplo en audio y videos porque son tareas vinculadas a la CPU en la que se ejecutan instrucciones que dependen de la velocidad del hardware.

● Un caso en el que usarías threads para resolver un problema y por qué.
R/ Un Thread también conocidos como sub-procesos o hilos es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Un thread es la unidad más pequeña a la cual un procesador puede asignar tiempo. Los hilos poseen la secuencia más pequeña de instrucciones a ejecutar, estos se crean, ejecutan y mueren dentro de los procesos, siendo capaces de compartir información entre ellos.

Caso: Utilizará threads para enviar requests HTTP porque son tareas relacionadas con usos externos (entrada/salida), Con los threads y los procesos seremos capaces de implementar la programación concurrente, y, dependiendo de la cantidad de procesadores la programación en paralelo.

● Un caso en el que usarías corrutinas para resolver un problema y por qué.
R/ Las corrutinas permiten escribir código asíncrono de forma secuencial, lo que reduce drásticamente la carga cognitiva, se usan porque son mucho más eficientes que los hilos. Varias corrutinas se pueden ejecutar utilizando el mismo hilo. Por tanto, mientras que el número de hilos que se pueden ejecutar en una aplicación es bastante limitado, se pueden lanzar tantas corrutinas como se necesite, el límite es casi infinito.

Caso: Un servidor web que tiene múltiples conexiones simultáneas, requiere programar la lectura y la escritura de todas ellas. Esto se puede implementar usando corrutinas, porque cada conexión es una corrutina que lee / escribe una pequeña cantidad de datos, luego "cede" el control al programador, que pasa a la siguiente corrutina (que hace lo mismo) a medida que recorremos todos los disponibles relación.

Optimización de recursos del sistema operativo

Si tuvieras 1.000.000 de elementos y tuvieras que consultar para cada uno de ellos información en una API HTTP. ¿Cómo lo harías? Explicar.
R/ Generaría una piscina de hilos para lanzar varias requests por hilo de manera concurrente, a medida que cada worker vaya resolviendo una tarea quedará disponible para ejecutar una nueva.

Análisis de complejidad

● **Dados 4 algoritmos A, B, C y D que cumplen la misma funcionalidad, con complejidades $O(n^2)$, $O(n^3)$, $O(2^n)$ y O (n log n), respectivamente, ¿Cuál de los algoritmos favorecerías y cuál descartarías en principio? Explicar por qué.**
R/ En primera instancia descartaria el algoritmo B ($O(n^3)$) y C(O(2n)) cúbico y exponencial respectivamente porque su complejidad tiende aumentar más rápidamente que los otros algoritmos.
Finalmente, tengo el algoritmo A ($O(n^2)$) y D (O (n log n)), cuadrático y logarítmico respectivamente, de los cuales escogería el logarítmico ya que su complejidad se mantiene casi linealmente.

● **Asume que dispones de dos bases de datos para utilizar en diferentes problemas a resolver. La primera llamada AlfaDB tiene una complejidad de O(1) en consulta y $O(n^2)$ en escritura. La segunda llamada BetaDB que tiene una complejidad de O(log n) tanto para consulta, como para escritura. ¿Describe en forma sucinta, qué casos de uso podrías atacar con cada una?**
R/
En el escenario AlfaDB se podría resolver un caso de sistema de gestion de creditos, puesto que en la consulta O(1) se limita a entregar una clave única de un crédito asignado y en escritura $O(n^2)$ involucra un ciclo de búsqueda de registros de créditos existentes antes de crear uno nuevo.

En el caso de BetaDB se podría resolver un caso de estudios estadísticos ya que en la consulta y escritura O(log n) las operaciones se limitan a condiciones particulares, por ejemplo rango de edades definidos, sexo, etc.