Enikio: Aplicación web para la búsqueda de apartamentos en renta cercanos a universidades en la ciudad de Cali

Camila Andrea Cardona Alzate
Ingeniería de Datos e Inteligencia Artificial
Universidad Autónoma de Occidente
Cali, Colombia
camila_and.cardona@uao.edu.co

Mariana Mera Gutierrez
Ingeniería de Datos e Inteligencia Artificial
Universidad Autónoma de Occidente
Cali, Colombia
mariana.mera@uao.edu.co

Maria Angélica Portocarrero Quintero
Ingeniería de Datos e Inteligencia Artificial
Universidad Autónoma de Occidente
Cali, Colombia
maria_a.portocarrero@uao.edu.co

Santiago Rodriguez Ramirez
Ingeniería de Datos e Inteligencia Artificial
Universidad Autónoma de Occidente
Cali, Colombia
santiago_rod.ramirez@uao.edu.co

Xilena Atenea Rojas Salazar
Ingeniería de Datos e Inteligencia Artificial
Universidad Autónoma de Occidente
Cali, Colombia
xilena_atenea.rojas@uao.edu.co

Dayanna Vanessa Suarez Mazuera
Ingeniería de Datos e Inteligencia Artificial
Universidad Autónoma de Occidente
Cali, Colombia
dayanna.suarez@uao.edu.co

***Abstract*— In this paper we describe the analysis and design of the solution Enikio. This solution is a web service which has an architecture based on microservices, and that is packed using docker compose. Also, the analysis and distribution of the dataset was done with Apache Spark and the results are shown in a dataset that is in the web service. We also evaluated other alternatives as a solution, and then chose one implementation, which is the one described in this text.**

Palabras clave—Enikio, Docker, Apache Spark, clúster, despliegue.

I. Introducción

Para estudiantes, profesores y colaboradores de las universidades de Cali, conocer personas que residen en lugares sumamente lejanos a la universidad a la que asisten es más común de lo que se piensa. Estas personas, usualmente presentan grandes problemas con el transporte, con el tiempo que requiere ir de sus hogares a la universidad, o con la inversión de dinero que este transporte y/o la alimentación dentro de la universidad conlleva. Ante esto, se ha evidenciado cómo muchas de las personas que se ven involucradas en esta problemática, empiezan una búsqueda de un nuevo hogar que les otorgue solución a sus necesidades, como lo es la comodidad de estar cerca a la universidad en la que trabajan o estudian. En este informe se explicará la solución implementada para esta problemática, teniendo en cuenta el uso de herramientas como Docker Compose y Apache Spark para montar la infraestructura de la aplicación.

II. Análisis de la aplicación

A. Descripción del dataset

La base de datos con la que trabaja la aplicación web incluye la información de los usuarios, apartamentos, postulaciones y universidades. Estas tablas permiten el inicio de sesión de los dueños de apartamentos y el administrador de la página web, así como mostrar las universidades y los apartamentos cercanos a estas. Los usuarios que busquen apartamento pueden postularse a estos con su información personal, y de esta interacción con la página sale el dataset objetivo. Este es el dataset de *postulaciones*, un archivo csv con la información del postulado como su *cédula*, su *ocupación*, y su *interés* (apartamento completo o habitación). También, está la información de la postulación en sí, como la *fecha* en la que se hizo, el *apartamento* al que el postulante aplicó y el *estado* de la solicitud.

La generación de esta tabla se hizo con ayuda Mockaroo, y Python. De la página generadora de datos se obtuvieron los datos de 10000 postulantes, y por medio de un código en python se simularon 20000 postulaciones de forma aleatoria a los más de 1000 apartamentos, que se recuperaron de la página web de Finca Raíz a través de web scraping.

B. Generación y selección de alternativas de solución

1. Uso de Kubernetes

Se planteó inicialmente el uso de Kubernetes para empaquetar el proyecto, pero se descartó casi al instante debido a su complejidad. Aunque Kubernetes es una plataforma muy poderosa, su configuración, administración y mantenimiento requieren conocimientos especializados. Además, se destaca en entornos con aplicaciones complejas, alta disponibilidad y escalabilidad, lo cual no es necesario para nuestro proyecto que es más pequeño en comparación a los proyectos que se pueden manejar con Kubernetes.

En cambio, optamos por Docker Compose, una opción más ligera y familiar para nuestro equipo. Esto nos permitió enfocarnos en el desarrollo del proyecto en lugar de aprender una herramienta nueva con capacidades que no son requeridas por el tamaño y alcance del proyecto.

2. Uso de Docker Swarm

Durante el diseño de la arquitectura se planteó también el uso de Docker Swarm. Se planeó disponer de un cluster de dos nodos, cuyo worker iba a alojar a dos instancias del servicio web “microweb”, sin embargo esta idea se descartó debido a la necesidad de mantener la simplicidad para el proyecto, se consideró más oportuno optar por una herramienta más sencilla e igualmente funcional.

3. Procesamiento de datos en streaming

En cuanto a la parte de procesamiento de datos, se propuso hacer procesamiento de datos en streaming utilizando Apache Spark. Se buscaba simular datos de postulaciones con una función aleatoria en un script de python, los cuales iban a ser enviados y analizados posteriormente por otro script de python que hiciera uso de Apache Spark Streaming para el procesamiento de los datos de las postulaciones en tiempo real, con el fin de obtener datos como las ocupaciones que hacen más postulaciones o las universidades cuyos apartamentos más cercanos reciben más postulaciones. Estos datos iban a ser presentados al usuario a través de un dashboard que se actualizara en tiempo real. Esta opción se descartó debido a que actualmente no se poseen los conocimientos para implementar un dashboard de este tipo.

C. Definición de la arquitectura completa del sistema y Pipeline

Fig. 1. Propuesta de pipeline.

La figura 2 muestra la propuesta de Pipeline para implementar la arquitectura escogida, el flujo de trabajo es el siguiente (de acuerdo a los números dentro del diagrama):

1. Se reúnen los datos que salen como resultado del webscraping de la página Fincaraiz, y la generación del dataset objetivo haciendo uso de Python y Mockaroo. Se envían a la base de datos MySQL, que está empaquetada en un contenedor de Docker.
2. Se transforman los datos de formato csv a json para hacer el procesamiento y distribución de los datos, haciendo uso de Apache Spark. Se actualiza la base de datos.
3. La base de datos se conecta con el servicio web para presentar el dashboard de los datos procesados con Apache Spark y en sí, atender las consultas que el usuario haga a la página.

El sistema se basa en una arquitectura de microservicios y API REST para implementar la aplicación, los microservicios son apartamentos, usuarios y postulaciones, y son empaquetados en un cluster de contenedores (teniendo cada uno de los microservicios un contenedor). Estos microservicios acceden a una base de datos relacional llamada “enikio” haciendo uso de MySQL, y de solicitudes HTTP para enviar, eliminar, modificar y recibir datos por medio de operaciones GET, POST, PUT y DELETE. Los datos fueron montados en un contenedor de mysql dentro del cluster, haciendo uso de un archivo init.sql para cargar los archivos csv con la información.

La arquitectura es implementada por medio de docker compose, lo que permite la escalabilidad de forma independiente y disponibilidad de cada uno de los servicios. Cada uno de los microservicios tienen sus funciones respectivas. El microservicio *usuarios* se encarga de gestionar la información de los usuarios arrendadores, postulados y administradores, al igual que validar sus credenciales. El microservicio de apartamentos gestiona la información de los apartamentos y universidades y encontrar los apartamentos más cercanos a cada universidad a través de sus coordenadas. Finalmente, el microservicio de postulaciones gestiona la información de las postulaciones a cada apartamento para que sean visualizadas por el arrendador y añadir a la tabla de usuarios la información de un postulado la primera vez que hace una postulación.

Los tres microservicios se exponen cada uno en un puerto diferente. El microservicio de *usuarios* se expone en el puerto 3001, el de *apartamentos* en el puerto 3002 y el de *postulaciones* en el puerto 3003. Además de los microservicios, dentro del cluster se encuentran otros tres servicios: microweb1, microweb2 y haproxy. Dentro de los servicios *microweb1 y microweb2* se incluye toda la parte del frontend de la aplicación, los archivos html y php con sus respectivas funciones. El servicio de *haproxy* se encarga del balanceo de carga de todos los servicios web, lo que permite manejar la disponibilidad y adaptar la aplicación dependiendo de la demanda dentro de la página web. Este funciona con planificación round robin, y teniendo en cuenta que la sesión php que se inicie perdure en el mismo servicio durante toda la navegación.

Para la parte del procesamiento de datos, se usó Apache Spark. Teniendo las tablas *aptos, universidades, postulaciones* y *usuarios* dentro de la base de datos, se importan en el código de procesamiento *enikiospark.py* y se pasan a formato json. Esto se logra con un cluster maestro de Spark y un worker. El worker paraleliza el cargado de los datos en dataframes para las postulaciones, los apartamentos y las universidades. Posteriormente, se aplican unas transformaciones a los datos para hallar en ellos los insights que le serán entregados al usuario. Para la entrega de los datos se logra mediante el siguiente procedimiento: se sobrescribe el archivo csv de universidades para agregar las nuevas columnas que nos proporcionan información valiosa sobre las universidades. Se crean dos archivos csv nuevos, *aptosPrecio.csv y occupationCount.csv* y luego se cargan a la base de datos a nuevas tablas. De esta manera es posible acceder a las nuevas tablas desde las rutas definidas en nuestra API; así se genera un dashboard en *analisisadmin.php* que hace un llamado a las métricas recién obtenidas para que el usuario pueda visualizarlas.

III. Diseño de la aplicación

A. Diagrama de los componentes a utilizar.

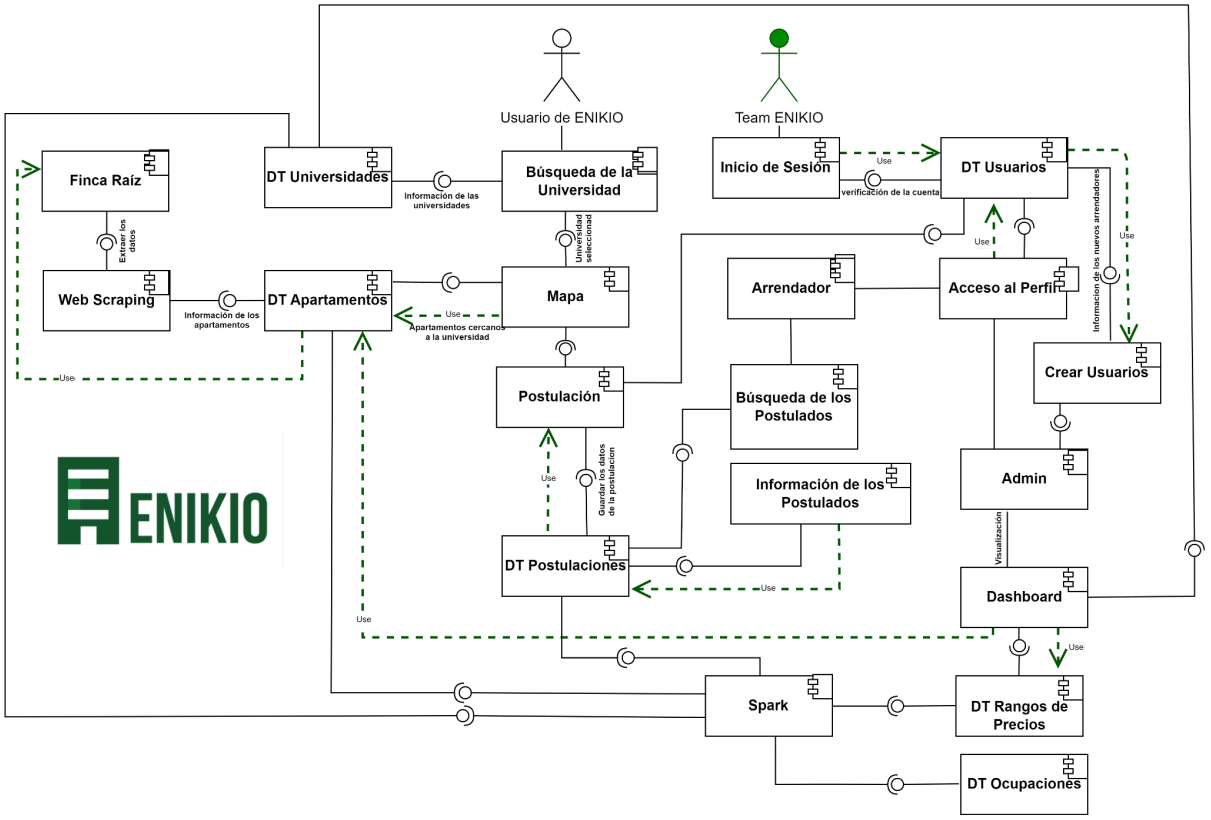

Fig. 2. Diagrama de Componentes

B. Relación y flujo de trabajo entre los componentes.

Relación entre los componentes (microservicios):

Usuarios y postulaciones: El usuario con el rol “admin” no puede visualizar las postulaciones de cada apartamento en particular, sin embargo, tiene una visión de la cantidad de postulaciones totales que hay en la base de datos de nuestra plataforma. El usuario de rol “arrendador” puede visualizar todas las postulaciones hechas a cada uno de sus apartamentos y, como implementación a futuro, cambiar el estado de la postulación que desee de “pendiente” a “aceptado” o “rechazado”, esto con el fin de llevar un orden entre sus propias postulaciones. Cuando se crea la postulación, al usuario se le solicitan los datos de documento de identidad (CC.), nombre, email, celular, ocupación (la cual puede elegir entre “docente”, “estudiante”, “colaborador” y “otra”) e interés (‘habitación’ o ‘apartamento completo’). Al ingresar estos datos, si la C.C. no coincide con una existente en la tabla, se crea un nuevo usuario, tomando los datos de C.C., nombre, email, celular y se le asigna el rol de “postulado”, con el fin de mostrar la información posteriormente en el área de postulaciones de la página de los arrendadores.

Usuarios y aptos: Todos los usuarios pueden visualizar los apartamentos, sin embargo la visualización varía dependiendo de su rol. El administrador puede ver todos los apartamentos en la plataforma, escogiendo una universidad y visualizando todos los apartamentos más cercanos a esta, el arrendador puede ver todos los apartamentos que sean de su propiedad y editar la cantidad de habitaciones disponibles y, por último, el usuario

que visite la plataforma puede visualizar los 100 apartamentos más cercanos a la universidad de su elección que tengan habitaciones disponibles.

Postulaciones y apartamentos: Cuando se va a generar una postulación, el formulario de postulación toma automáticamente el id del apartamento seleccionado en el mapa para ingresarlo en ese registro de la tabla de postulaciones, relacionando cada postulación con el respectivo apartamento. Por otro lado, el usuario arrendador puede visualizar las postulaciones de cada uno de sus apartamentos en concreto haciendo clic en el botón junto a la información del apartamento.

Flujo de trabajo entre los componentes:

El proceso comienza con la entrada de los usuarios, quienes buscan la universidad en la que necesitan el apartamento. A través de una conexión con la base de datos de Enikio, se realiza una consulta en la tabla de universidades para obtener información relevante. Posteriormente, se muestra un mapa interactivo que presenta los apartamentos más cercanos a la universidad seleccionada. La obtención de esta información se realiza mediante el web scraping de la plataforma Finca raíz, donde se extraen datos y se guardan en la tabla de apartamentos. Una vez que el usuario ha seleccionado un apartamento de su interés, se le solicitan algunos datos personales para completar la postulación. Esta información se almacena en la base de datos en la tabla de postulaciones para su posterior procesamiento y seguimiento.

Por otra parte, si son usuarios arrendadores y administradores, deben iniciar sesión para acceder a la plataforma. Siendo arrendadores, tienen la capacidad de buscar y visualizar los postulados mediante consultas en la tabla de postulaciones. Estas consultas les permiten acceder a la información relevante sobre los postulados y evaluar las solicitudes recibidas de manera efectiva. Por otro lado, los administradores tienen acceso a un dashboard que se conecta a la base de datos. Este dashboard proporciona análisis detallados sobre los apartamentos y universidades registrados en la plataforma. Además, los administradores tienen la capacidad de crear usuarios arrendadores, lo que implica la generación de registros en la tabla de usuarios. Este proceso permite una gestión eficiente de los arrendadores, brindándoles acceso adecuado y manteniendo un seguimiento completo de los usuarios del sistema.

El dashboard ya mencionado se conecta con la base de datos, que tiene los resultados de un procesamiento distribuído que se hizo con Apache Spark. A través del script de python enikiospark.py, se crea una sesión de Spark, donde se establecen las funciones, las transformaciones de los datos y el cálculo de estadísticas y agregaciones. A partir de este análisis se crean los dos componentes de DT Rangos de Precio y DT ocupaciones, que son dos tablas nuevas con los resultados. Además de esto se añaden dos nuevas columnas a la tabla de universidades, por lo que el componente Spark envía nuevos datos a este otro componente.

C. Descripción de los componentes del sistema.

Usuarios: Este componente se encarga de gestionar la información de los usuarios arrendadores, postulados y administradores, al igual que validar sus credenciales, haciendo uso de la tabla de usuarios.

Aptos: Este componente gestiona la información de los apartamentos y universidades, al igual de encargarse de encontrar los apartamentos más cercanos a cada universidad a través de sus coordenadas. Hace uso de las tablas de apartamentos y universidades.

Postulaciones: Este componente se encarga de gestionar la información de las postulaciones a cada apartamento para que sean visualizadas por el arrendador y añade a la tabla de usuarios la información de un postulado la primera vez que hace una postulación. Este componente hace uso de las tablas de postulaciones y usuarios.

Base de datos: En este componente se encuentra la información de las universidades, usuarios, apartamentos y postulaciones.

Mapa: Este componente permite ver los apartamentos cercanos a la universidad seleccionada por el usuario y realizar una postulación del apartamento de su interés. El mapa está conectado con los datos de las universidades y apartamentos, previamente extraídos de la plataforma de Finca raíz.

Finca Raíz: Este componente se utilizó para extraer los datos de los apartamentos por medio de web scraping.

Dashboard: Este componente lo puede visualizar el administrador a través del servicio web donde puede observar el total de apartamentos, los aptos cerca de cada universidad, el número de las postulaciones realizadas, el número de aptos con dos habitaciones disponibles y de arrendadores.

Spark: Este componente lleva a cabo varios análisis distribuídos de los datos, incluyendo la agrupación y el cálculo del promedio de los precios de los apartamentos por universidad, el conteo de las postulaciones y las habitaciones por rango de precio y por universidad.

D. Diagrama de despliegue.

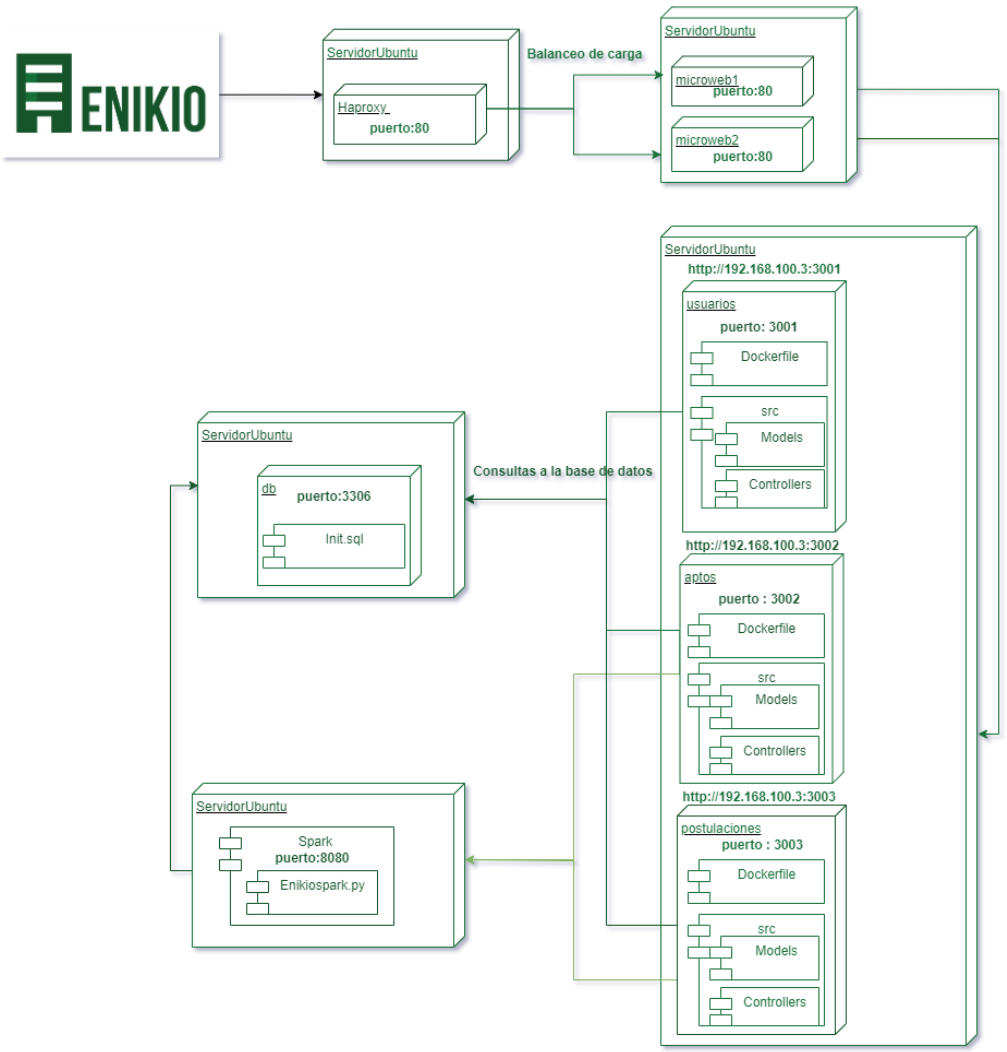

Fig. 3. Diagrama de despliegue.

En este diagrama se puede ver cómo se implementó la solución dentro de un solo master que es *servidorUbuntu*. Empezando por la parte del frontend, se implementó un balanceo de carga con Haproxy, que escucha del puerto 80, y que redirecciona ya sea a microweb1 o microweb2, ambos configurados en el Haproxy exponiendo en el puerto 80. Dentro del despliegue están los tres microservicios, usuarios (puerto 3001), aptos (3002) y postulaciones (3003), cada uno con su respectivo Dockerfile para el levantamiento de sus imágenes y su código fuente dentro de la carpeta src. Los microservicios de aptos y postulaciones tienen sus respectivas tablas, que son tomadas para hacer el análisis distribuído de Spark.

Este procesamiento de spark se hace a través del script enikiospark.py que hace lo siguiente:

1. Importación de bibliotecas: Se importan las bibliotecas necesarias, como `requests`, `pyspark.sql`, `pyspark.sql.functions`, `pyspark.sql.types`, `math`, `json` y `sys`.

2. Creación de la sesión de Spark: Se crea una instancia de `SparkSession` con el nombre de la aplicación "Enikio EDA".

3. Definición de funciones:
   - `haversine`: Función que calcula la distancia entre dos puntos de una esfera utilizando la fórmula de haversine.
   - `nearest_university`: Función que encuentra la universidad más cercana a partir de una lista de distancias.

4. Creación de User Defined Functions (UDFs): Se definen dos UDFs utilizando las funciones anteriormente definidas.

5. Carga de datos:
   - Se realizan solicitudes HTTP utilizando la biblioteca `requests` para obtener datos de tres endpoints.
   - Los datos JSON obtenidos se convierten en listas de strings JSON.
   - Las listas de strings JSON se convierten en Data Frames utilizando el método `read.json`.

6. Transformaciones de datos:
   - Se realizan transformaciones en los Data Frames para obtener las columnas de coordenadas separadas.
   - Se seleccionan las columnas necesarias en los DataFrames.

7. Cálculo de distancias:
   - Se itera sobre las filas del DataFrame de universidades para calcular la distancia a cada universidad y agregar las columnas correspondientes al Data Frame de apartamentos.
   - Se crea una lista de columnas de distancia.
   - Se agrega una columna de universidad más cercana y una columna de distancia a la universidad más cercana al Data Frame de apartamentos.
   - Se eliminan las columnas de distancia individuales.

8. Cálculo de estadísticas y agregaciones:
   - Se calculan los cuartiles y el promedio de los precios en el DataFrame de apartamentos.
   - Se agrega una columna de rango de precio al Data Frame de apartamentos.
   - Se realizan agrupaciones y cálculos de promedio y recuento en los Data Frames de apartamentos y universidades.

9. Unión de DataFrames:
   - Se unen los DataFrames de universidades y estadísticas utilizando las columnas "nombre" y "nearest_university".

- Se realizan algunas transformaciones y renombramientos de columnas.

10. Impresión y escritura de resultados:
    - Se imprimen en la consola los resultados de los DataFrames de recuento de ocupación, universidades y rangos de precio de apartamentos.
    - Se convierten los DataFrames a Pandas DataFrames.
    - Los Pandas DataFrames se guardan como archivos CSV.

11. Detención de la sesión de Spark.

A partir de los resultados de este procesamiento de datos, se actualiza el servicio db. Se generan dos nuevas tablas con los resultados de los análisis (que serán consultados por los microservicios para mostrar el dashboard desde el usuario administrador), y se actualiza la información de las universidades al añadir dos nuevas columnas. La base de datos fue implementada haciendo uso de una imagen mysql 5.7 que recibe información a través del puerto 3306.

V. Pruebas de carga y escalabilidad

Para las pruebas de carga y balanceo se implementó Jmeter. Para el primer escenario, sin hacer escalabilidad de ninguno de los servicios se creó un grupo de usuarios con las siguientes configuraciones: 1000 usuarios haciendo peticiones en un conteo de cinco segundos con un loop de 10. Los resultados son los siguientes:

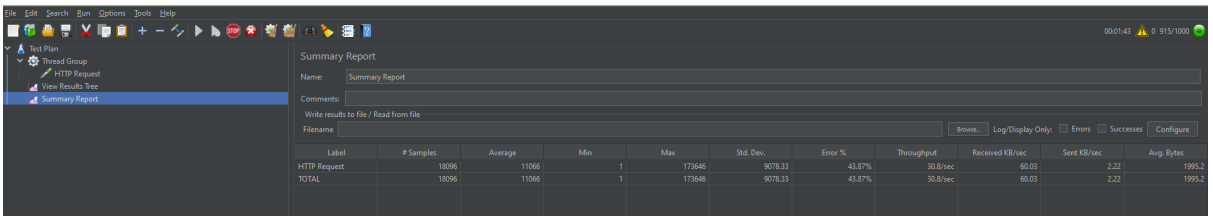

Fig. 4. Pruebas de carga I.

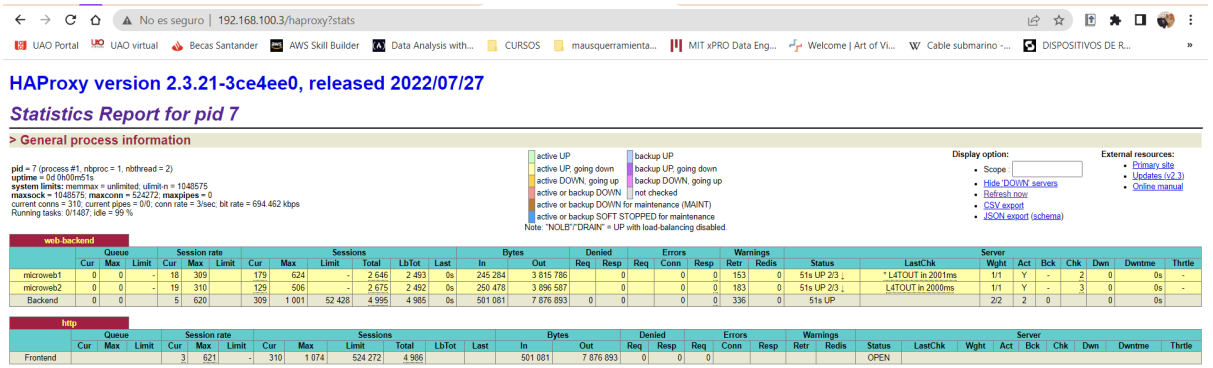

Fig. 5. Estadísticas desde Haproxy I

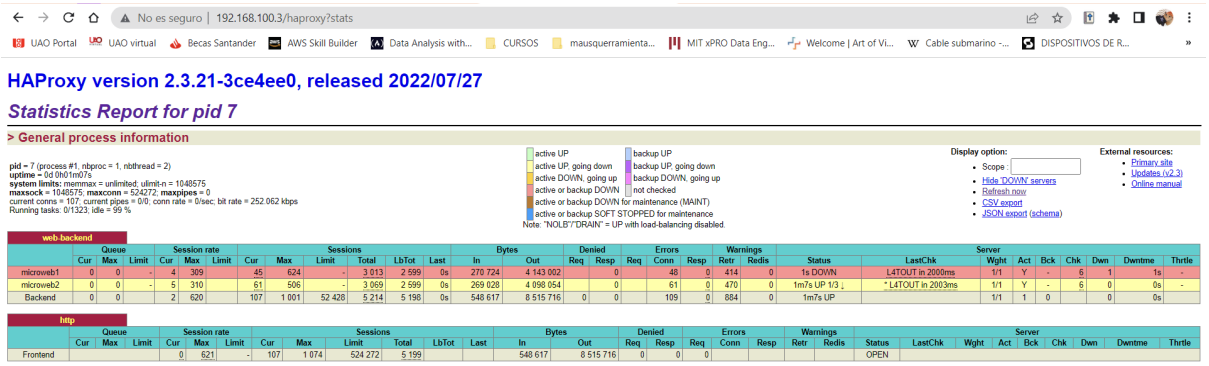

Fig. 6. Estadísticas desde Haproxy II.

Las estadísticas arrojan un desempeño regular, con más del 43% de error, y en Haproxy se comprueba que funciona el balanceo de carga, sin embargo se ve cómo los servicios empiezan a caerse y se demoran un tiempo considerable en volver a funcionar. A la hora de hacer el seguimiento en tiempo real de los HTTP request se observan varias peticiones que son interrumpidas. Para mejorar este comportamiento de las peticiones se escalaron los servicios microweb1 y microweb2, cada uno con 4 réplicas. Después de hacer esta acción, las estadísticas son las siguientes:

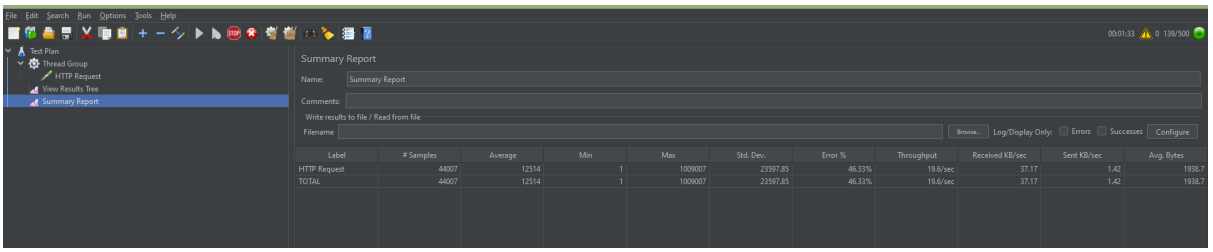

Fig. 7. Pruebas de carga II.

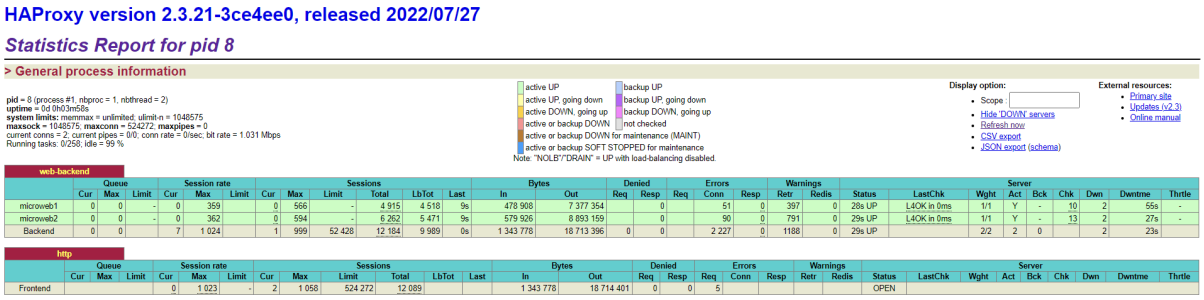

Fig. 8. Estadísticas desde Haproxy III**.**

Para esta implementación sale el mismo porcentaje de error,sin embargo las peticiones que se caen se recuperan casi al instante al revisar las HTTP Request. En este caso los servicios permanecen estables mientras se hacen las peticiones, por lo que se puede concluir que el escalar los servicios funciona para la carga que se tenía prevista, teniendo en cuenta el aproximado de peticiones que se podrían recibir en Cali y sus alrededores.

VI. Conclusiones

Después de la implementación de la solución con la arquitectura y diseño propuesto, se plantean las siguientes conclusiones:

1. El uso de una arquitectura de microservicios y API REST permite una mayor modularidad y flexibilidad en el desarrollo de la aplicación.

2. La inclusión del servicio Haproxy en el cluster permite distribuir la carga de trabajo del servicio web y garantizar la disponibilidad de la aplicación.

3. La integración de Apache Spark en el proyecto permite realizar un procesamiento eficiente de grandes volúmenes de datos, esto permite obtener insights y métricas relevantes.

4. El uso de contenedores, empaquetados en un cluster mediante Docker Compose facilita la portabilidad y la escalabilidad de los microservicios.