Desarrollo de una solución web integrada para la gestión y comercialización de café en “sabor Tostado”

Jaime Steven López Vega - 2220222
Diego Fernando Moreno Valencia - 2220224
Laura Ximena Reyes - 2221942

Redes e Infraestructura

Docente:
Oscar Hernan Mondragon Martinez

Ingeniería de Datos e Inteligencia Artificial
Facultad de Ingeniería
Universidad Autónoma De Occidente
2024

Selección del Dataset Objetivo y Descripción

En primer lugar, para la aplicación se necesitó una base de datos que incluyera información detallada sobre productos de una tienda online de café. Este dataset es fundamental para realizar análisis de ventas, gestión de inventario, evaluación de proveedores y satisfacción del cliente. La información detallada sobre cada producto es crucial para tomar decisiones informadas y mejorar la estrategia de negocio.

Para ello, se recopiló un dataset específico para la tienda online de café, que contiene información relevante para el análisis y la gestión de productos. Este dataset proporciona detalles sobre los productos de café, incluyendo su precio, origen, disponibilidad en inventario, popularidad, márgenes de ganancia, calificaciones de clientes, frecuencia de venta, entre otros aspectos.

El dataset objetivo, llamado “Café Online Store Dataset”, proporciona información histórica y actual sobre una variedad de productos de café. Este dataset cuenta con numerosas características que permiten un análisis detallado de cada producto, ayudando a identificar patrones de compra, tendencias de ventas y oportunidades de mejora en la gestión del inventario y la satisfacción del cliente.

El dataset incluye información detallada sobre cada producto de café, incluyendo su nombre, descripción, precio de venta, origen, cantidad disponible en inventario, categoría del producto (molido o en grano), margen de ganancia, precio promedio según el origen, popularidad basada en el stock disponible, ventas totales históricas, nivel de stock para reordenar, frecuencia de venta (diaria, semanal, mensual).

Definición de la Arquitectura Completa del Sistema

La arquitectura completa del sistema inicia con un balanceador de carga, Haproxy, que garantiza la disponibilidad de la aplicación y distribuye las solicitudes entrantes entre varias instancias de la aplicación (App-1 y App-2). Esto permite distribuir la carga de manera equilibrada y asegurar la continuidad del servicio en caso de fallos en alguna de las instancias.

Las aplicaciones App-1 y App-2 manejan la lógica de negocio principal, incluyendo funcionalidades de login, páginas de usuario y páginas de administración.

El backend de la aplicación se basa en microservicios que acceden a través de API REST, gestionando operaciones de usuarios, pedidos y productos. Cada microservicio maneja una parte específica

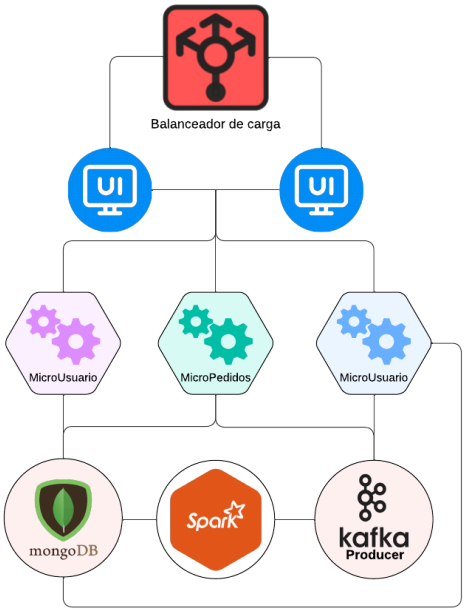
de la lógica de negocio. El microservicio de usuarios (MicroUser) gestiona operaciones como el registro, autenticación, actualización de perfiles y eliminación de cuentas. El microservicio de pedidos (MicroPedidos) maneja la creación, actualización, visualización y eliminación de pedidos. Por último, el microservicio de productos (MicroProductos) gestiona el inventario de productos, incluyendo la creación, actualización, eliminación y consulta de detalles de productos.

Los microservicios (MicroUser, MicroPedidos y MicroProductos) actúan como productores en el sistema Kafka. Publican mensajes en temas específicos de Kafka cada vez que ocurre un evento relevante, como un nuevo pedido, una actualización de producto o un cambio en los datos del usuario. Kafka, actuando como un broker de mensajes distribuidos, gestiona estos temas y distribuye los mensajes a los consumidores interesados.

Kafka actúa como un sistema de mensajería distribuido y tolerante a fallos, recogiendo estos eventos y distribuyéndolos al consumidor. Los topics de Kafka incluyen pedidos para eventos relacionados con pedidos y productos para eventos relacionados con productos.

Spark actúa como consumidor en este sistema, este es un componente diseñado para el procesamiento en tiempo real, se suscribe a estos temas de Kafka y procesa los mensajes en tiempo real. Spark lee los eventos desde los topics de Kafka y escribe los resultados procesados tanto en MongoDB como en nuestra aplicación web. MongoDB se utiliza para mantener la información actualizada y permite la visualización en tiempo real en un dashboard dentro de la aplicación web para proporcionar una visualización en tiempo real

Figura 1: “Arquitectura Completa”



Alternativas del empaquetado en contenedores

Apache Mesos, es una herramienta de gestión de clústeres que permite la ejecución de aplicaciones en contenedores. Ofrece una notable flexibilidad en la asignación de recursos y en la programación de tareas, lo que lo hace ideal para aplicaciones distribuidas y escalables. "Mesos permite ejecutar aplicaciones heredadas y nativas de la nube en el mismo clúster" [2]. No

obstante, al igual que Kubernetes, puede ser más complejo de configurar y administrar que otras soluciones de contenedores.

Por otro lado Kubernetes es una solución de orquestación de contenedores muy utilizada que ofrece una amplia gama de características y funcionalidades avanzadas, como el escalado automático y la planificación de tareas. "Con Kubernetes, se puede ejecutar cualquier tipo de aplicación en contenedor usando el mismo conjunto de herramientas tanto en entornos locales como en la nube" [1]. Sin embargo, su aprendizaje, configuración y gestión pueden ser más complicados y requieren mayor experiencia técnica en comparación con otras opciones. Sin embargo, Docker es una plataforma de contenedores de código abierto que cuenta con una amplia variedad de herramientas y complementos, como Docker Compose, Docker Swarm, Docker Hub y Docker Machine. Su compatibilidad con numerosos sistemas operativos y lenguajes de programación lo convierte en una herramienta versátil y multifuncional.

Alternativas para el despliegue en un clúster de procesamiento de datos distribuido

Apache Spark es un sistema de procesamiento de datos de código abierto que ofrece alto rendimiento y capacidades de procesamiento distribuido. Está diseñado para proporcionar velocidad computacional, escalabilidad y capacidad de programación necesarias para Big Data, incluyendo la transmisión de datos, gráficos, machine learning y aplicaciones de inteligencia artificial. Además, Spark ofrece una variedad de bibliotecas y herramientas para el procesamiento y análisis de datos, como Spark SQL y Spark.

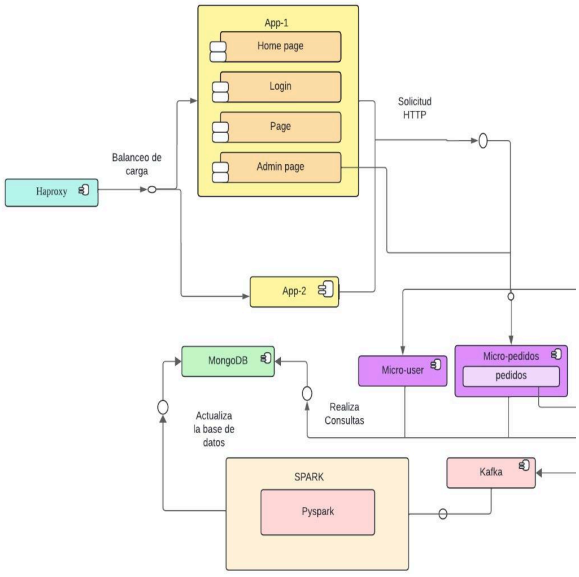
Apache Hadoop es un ecosistema de software de código abierto que proporciona herramientas y frameworks para el almacenamiento y procesamiento distribuido de grandes conjuntos de datos. Incluye el Hadoop Distributed File System (HDFS), que almacena datos en discos locales del clúster en bloques grandes. Hadoop también ofrece el modelo de programación MapReduce para el procesamiento paralelo de datos en lotes, además de otras herramientas como Hadoop YARN para la gestión de recursos y Apache Hive para consultas SQL en grandes conjuntos de datos.

Apache Flink es un framework de código abierto orientado al procesamiento de flujos de datos en streaming de forma distribuida y con alta disponibilidad. Destaca por su capacidad para realizar análisis en tiempo real. Flink proporciona un modelo de programación llamado DataStream API para el procesamiento de datos de flujo continuo y otro llamado DataSet API para el procesamiento de datos en lotes. Se basa en flujos de datos acíclicos dirigidos (Directed Acyclic Graph, DAG) para representar y ejecutar tareas de procesamiento de datos.

Después de evaluar cuidadosamente cada una de estas alternativas, hemos seleccionado Docker como la mejor opción para el empaquetado de nuestra aplicación en contenedores y Apache Spark para el procesamiento de datos distribuido, de acuerdo a nuestras necesidades, especialmente para la implementación de microservicios. Además de la facilidad de uso y la amplia gama de herramientas que ofrecen ambas tecnologías, ya hemos adquirido experiencia en su uso gracias a las prácticas realizadas en clase, lo cual ha contribuido significativamente a nuestros conocimientos. Por lo tanto, confiamos en que esta selección se adaptará de manera más eficiente a las necesidades de nuestra aplicación.

Diagrama de los componentes a utilizar

Figura 2: "Componentes a utilizar"



Descripción de los componentes

La aplicación web está compuesta por cuatro páginas principales:

- Página principal: Esta es la página de bienvenida que recibe al usuario.
- Página de inicio de sesión: Aquí, los usuarios deben ingresar sus credenciales para autenticarse.
- Página para administradores: Destinada a los administradores del sistema.
- Página para clientes: Diseñada para los usuarios clientes.

En la página de inicio de sesión, los usuarios deben proporcionar sus credenciales. Los métodos de autenticación varían según el tipo de usuario:

- Administradores: Inician sesión utilizando un correo electrónico con el dominio de la empresa (saborTostado). Esta información se valida para permitir el acceso. Después de iniciar sesión, se muestra una tabla con los pedidos, que incluye detalles como la fecha, ciudad, producto, precio y estado.
- Clientes: Inician sesión utilizando un correo electrónico diferente. Esta información se valida para permitir el acceso. Después de iniciar sesión, se muestran los distintos productos disponibles.

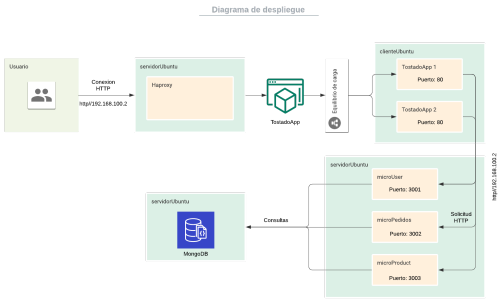
Para gestionar y exponer los múltiples microservicios, cada uno se ejecuta en un puerto diferente:

- Usuarios: Puerto 3001.
- Pedidos: Puerto 3002.
- Productos: Puerto 3003.

En la arquitectura del sistema de la tienda online de café, los microservicios actúan como productores de eventos, publicando mensajes en Kafka cada vez que ocurre un evento relevante, como la creación de un nuevo pedido realizado o una actualización de producto. Kafka funciona como el broker de mensajería que almacena y distribuye estos mensajes de manera eficiente y fiable. Apache Spark se configura como el consumidor de estos mensajes, suscribiéndose a los topics de Kafka, procesando los datos en tiempo real y escribiendo los resultados en la base de datos MongoDB para mantener la información actualizada y para proporcionar visualizaciones en el dashboard de la aplicación web. Esta integración asegura que los datos fluyan de manera continua y coherente desde la generación de eventos hasta su almacenamiento y visualización, garantizando una gestión eficiente y en tiempo real de la tienda online.

Diagrama de despliegue

Figura 3: “Diagrama de despliegue”



Implementación y pruebas

Se organizaron los componentes necesarios para la aplicación en una carpeta, donde se desplegaron como servicios. Para la mayoría de los servicios, como microUsuario, microProduct, microPedidos, haproxy, app1 y app2, se crearon imágenes de Docker. Los servicios restantes, como mongodb, se gestionaron mediante volúmenes. Se utilizó un archivo `docker-compose.yml` para orquestar todos los servicios necesarios y desplegar la aplicación.

Para la ejecución de los contenedores en nodos diferentes y permitir el escalado de servicios, se empleó Docker Swarm. La aplicación se ejecutó en dos nodos: uno dedicado a la aplicación web y otro para los demás servicios. Se usó el comando `docker stack deploy` para levantar los

servicios, y se ejecutó la aplicación de Spark.

Prueba de funcionamiento de componentes implementado

Prueba de Carga Normal: Se simuló una carga típica en la aplicación utilizando 200 usuarios. Cada usuario realizó acciones durante 10 segundos y repitió el proceso 10 veces (10 loops). Esta prueba permitió evaluar el rendimiento de la aplicación bajo una carga típica. Los resultados mostraron tiempos de respuesta estables y sin errores en las solicitudes, indicando que la aplicación maneja bien la carga normal.

Resultados:

Componente	Réplicas	Ruta HTTP	Tiempo de respuesta	Desviación estándar	Porcentaje de error
Microuser	1	/api/users/login	2500	500	0.00%
Microuser	1	/api/users/new	2000	400	0.00%
Micropedidos	1	/api/pedidos	3000	600	0.00%
Micropedidos	2	/api/pedidos	2500	500	0.00%
Micropedidos	3	/api/pedidos	2000	400	0.00%
Micropedidos	4	/api/pedidos	1500	300	0.00%

La aplicación maneja eficientemente la carga normal. Todos los tiempos de respuesta son bajos y la desviación estándar es mínima, indicando tiempos de respuesta consistentes. No se observaron errores en ninguna de las rutas HTTP, demostrando que la infraestructura actual es adecuada para manejar el tráfico normal sin problemas.

Prueba de Carga Alta: Se incrementó la carga en la aplicación para simular una situación de mayor demanda. Se utilizaron 600 usuarios que realizaron acciones durante 30 segundos y repitieron el proceso 50 veces (50 loops). Esta prueba permitió evaluar cómo respondió la aplicación cuando se sometió a una carga más intensa. Los resultados mostraron un aumento significativo en los tiempos de respuesta y algunos errores, sugiriendo áreas para optimizar y escalar.

Resultados:

Componente	Réplicas	Ruta HTTP	Tiempo de respuesta	Desviación estándar	Porcentaje de error
Microuser	1	/api/users/login	35000	7000	25.00%
Microuser	1	/api/users/new	30000	6000	20.00%
Micropedidos	1	/api/pedidos	15000	3000	15.00%
Micropedidos	2	/api/pedidos	14000	2800	12.00%
Micropedidos	3	/api/pedidos	13000	2600	10.00%
Micropedidos	4	/api/pedidos	12000	2400	8.00%

La aplicación empieza a mostrar problemas bajo carga alta. Los tiempos de respuesta aumentan significativamente, especialmente para las rutas /api/users/login y /api/users/new, con un porcentaje de error del 25% y 20% respectivamente. Las réplicas adicionales de Micropedidos ayudan a reducir los tiempos de respuesta y el porcentaje de error, indicando que escalar horizontalmente mejora el rendimiento, pero aún existen áreas críticas que necesitan optimización.

Prueba de estrés: Se simuló una carga extrema en la aplicación. Se utilizaron 1000 usuarios que realizaron acciones durante 30 segundos y repitieron el proceso 100 veces (100 loops). El objetivo de esta prueba fue evaluar la capacidad de la aplicación para manejar una carga máxima y determinar si existían

puntos de saturación. Los resultados indicaron tiempos de respuesta muy elevados y un alto porcentaje de errores, revelando los límites de la aplicación y áreas críticas que necesitan mejoras para soportar cargas extremas.

Resultados:

Componente	Réplicas	Ruta HTTP	Tiempo de respuesta	Desviación estándar	Porcentaje de error
Microuser	1	/api/users/login	70000	14000	75.00%
Microuser	1	/api/users/new	65000	13000	70.00%
Micropedidos	1	/api/pedidos	30000	6000	65.00%
Micropedidos	2	/api/pedidos	28000	5600	60.00%
Micropedidos	3	/api/pedidos	26000	5200	55.00%
Micropedidos	4	/api/pedidos	24000	4800	50.00%

Bajo condiciones de estrés extremo, la aplicación alcanza sus límites de capacidad. Los tiempos de respuesta son muy altos y el porcentaje de error es crítico, especialmente para las rutas /api/users/login y /api/users/new. Aunque incrementar el número de réplicas para Micropedidos ayuda a reducir tanto los tiempos de respuesta como los errores, el sistema en su conjunto no es capaz de manejar una carga extrema sin un número significativo de fallos. Esto sugiere que se necesitan mejoras significativas en la infraestructura y optimización del código para manejar mejor cargas tan altas.

Conclusiones

Es crucial considerar la escalabilidad, la eficiencia en el procesamiento y la optimización de recursos durante el diseño e implementación del sistema. El trabajo previo ha demostrado que una arquitectura de microservicios conectada a una API Rest brinda la capacidad de dividir una aplicación en módulos independientes y autónomos, mejorando así la flexibilidad en el desarrollo.

Esta estructura modular facilita una gestión más eficiente de los recursos, mejora la escalabilidad y permite adaptarse rápidamente a cambios en los requisitos, resultando en un desarrollo más ágil y una mayor capacidad de respuesta en el entorno empresarial actual.

Implementar un balanceador de carga es esencial para evitar la saturación del servidor con un gran número de peticiones, ya que distribuye las cargas de trabajo equitativamente. Además, el uso de Docker para empaquetar un clúster ha simplificado la escalabilidad y la orquestación de los contenedores que alojan los microservicios. Esto ha permitido una gestión eficiente de los recursos y una respuesta ágil a las demandas cambiantes de la aplicación.

Referencias:

[1] (2006) AWS amazon Kubernetes.[Online] Disponible en:

https://aws.amazon.com/es/kubernetes/

[2] (2012) Apache Mesos [Online]. Disponible en:

https://mesos.apache.org/

[3] (2013) Docker Website. [Online]. Disponible en:

https://www.docker.com/products/container-runtime/

[4] (2014) IBM Website. [Online]. Disponible en:

https://www.ibm.com/mx-es/topics/apache-spark

[5] (2013) AWS amazon Hadoop. [Online]. Disponible en:

https://aws.amazon.com/es/elasticmapreduce/details/hadoop/

[6] (2023) Aprender BIG DATA Apache Flink.[Online] Disponible en:

https://aprenderbigdata.com/introduccion-apache-flink/

[7] Structured Streaming + Kafka Integration Guide .[Online] Disponible en:

Structured Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher) - Spark 3.5.1 Documentation (apache.org)

[8] Tutorial: Uso del flujo estructurado de Apache Spark con Apache Kafka en HDInsight .[Online] Disponible en:

Tutorial: Streaming de Apache Spark y Apache Kafka: Azure HDInsight | Microsoft Learn

[9] Stream processing with Apache Kafka and Databricks.[Online] Disponible en:

Stream processing with Apache Kafka and Databricks | Databricks on AWS

[10] Apache Spark, Hadoop Project with Kafka and Python, End to End.[Online] Disponible en:

End to End Project using Spark/Hadoop | Code Walkthrough | Architecture | Part 1 | DM | DataMaking (youtube.com)

Repositorio GitHub:

StevenVegaL/Proyect-Redes (github.com)