Proyecto de curso - segunda parte

Juan Carlos Quintero, Camilo Delgado, Juan Andrés López, Dilian Mayerly
Universidad Autónoma de occidente
Cali, Colombia

Selección dataset objetivo

Según el Centro Nacional de Consultoría, el indicador de transformación digital de las empresas colombianas es de menos del 1%, lo cual es una cifra bastante preocupante si tenemos en cuenta que cerca del 71% de los pagos se realiza de manera digital, ya sea por medio de las pasarelas de comercio electrónico, los pagos por PSE, las páginas web de entidades financieras, los códigos QR, los datáfonos y los monederos digitales.

Por ende, el servicio web que se propone en este trabajo tiene como objetivo facilitar la digitalización de los negocios, ofreciéndoles una plataforma donde pueden registrar y controlar su inventario y ventas de forma fácil y segura. Además, les permite crear su propia tienda virtual, donde sus clientes pueden acceder a sus productos actuales, consultar información sobre ellos y finalmente comprarlos sin salir de casa. De esta manera, pueden participar del dinámico mercado del comercio electrónico, que en Colombia representa el 71% de los pagos digitales y el 67.1% de las compras en línea.
El dataset que se utiliza para el servicio web es un conjunto de datos de venta de celulares entre los años 2014 y 2015, que incluye información sobre el historial de compras, los vendedores y las especificaciones de los productos.

Registro de ventas: contiene cerca de 10.000 registros de las compras realizadas por los clientes, con cuatro variables: fecha, representante (que identifica al vendedor), código del producto y cantidad comprada.
Información de los clientes: contiene el nombre de 73 clientes los cuales serán utilizados posteriormente para el microservicio de usuarios, simulando una plataforma real.
Información de los productos: contiene 12 productos diferentes, con sus características técnicas y comerciales, como código, descripción, precio de venta, disponibilidad (en stock o agotado) y cantidad vendida.
Sin embargo, lo más importante no es el dataset en sí, sino cómo se procesa, analiza y visualiza la información para generar insights, recomendaciones y soluciones que ayuden a los usuarios a tomar mejores decisiones. Por eso, en este trabajo se mostrará cómo se puede utilizar el servicio web para obtener información valiosa sobre el mercado, los clientes, los productos y los vendedores, y cómo se puede mejorar la experiencia de compra y venta mediante el uso de herramientas digitales.

Link del kaggle donde obtuvimos el dataset: https://www.kaggle.com/datasets/papasricas/datos-de-venta-de-tiendas-en-peru10midatos

DEFINICIÓN DE LA ARQUITECTURA COMPLETA DEL SISTEMA

La arquitectura del sistema se basa en el paradigma de microservicios, que consiste en descomponer la aplicación en unidades funcionales independientes y autónomas, que se comunican entre sí mediante API REST. Una API REST es una interfaz de programación de aplicaciones que sigue los principios del estilo arquitectónico REST (Representational State Transfer), que permite el intercambio de datos entre diferentes sistemas de forma estandarizada y eficiente.

Los beneficios de usar una arquitectura basada en microservicios y API REST son:
- Mayor escalabilidad y rendimiento, ya que se puede aumentar o disminuir el número de instancias de cada microservicio según la demanda, y se puede distribuir la carga entre diferentes servidores o nodos.
- Mayor flexibilidad y agilidad, ya que se puede desarrollar, probar e implementar cada microservicio de forma independiente, sin afectar al resto del sistema, y se puede usar diferentes tecnologías o lenguajes de programación para cada uno.
- Mayor fiabilidad y disponibilidad, ya que se reduce el acoplamiento entre los componentes del sistema, y se puede aplicar técnicas de tolerancia a fallos, como el balanceo de carga, el circuit breaker o el fallback.
- Mayor facilidad de mantenimiento y evolución, ya que se puede actualizar o reemplazar cada microservicio sin necesidad de parar el sistema completo, y se puede modularizar y reutilizar el código.

Componentes de la arquitectura definida y relaciones entre ellos:
- Microservicio de Usuarios (MicroUsuarios):
  Será el encargado de gestionar todas las solicitudes respecto a la creación y edición de los usuarios.
- Microservicio de Productos (MicroProductos):
  Será el encargado de almacenar toda la información sobre los productos que se van a vender (teléfonos). Se podrá acceder a ellos para ser mostrada a los clientes, cuando se requiera modificar por el vendedor (root) o por el microservicio de ventas necesite hacer tareas en él.
- Microservicio de Ventas (MicroVentas):
  Será el encargado de almacenar toda la información sobre

las ventas que se vayan realizando, verificando si existe disponibilidad con el microservicio de productos, editando la cantidad en stock y generando un nuevo registro en la base de datos de ventas.
La arquitectura propuesta permitirá una alta escalabilidad y flexibilidad, ya que cada microservicio puede ser desarrollado, implementado y escalado independientemente de los demás. Además, la API REST permitirá una fácil integración con otras aplicaciones y sistemas externos

Para realizar el análisis distribuido a partir del dataset obtenido, se hizo la implementación de Apache Spark ya que esta dispone de la facilidad de procesar datos en paralelo y está hecha para leer grandes volúmenes de datos. Fue utilizado para realizar las estadísticas en los microservicios de aerolíneas y aeropuertos.

Ahora bien, para generar un avance en lo que se tiene planeado, se necesita el uso de una tecnología para empaquetar la aplicación en contenedores sobre el dataset seleccionado.

Al evaluar diferentes alternativas de solución para el empaquetado de la aplicación en contenedores se consideraron 2 opciones, Docker y Kubernetes. Y para el despliegue en un cluster de procesamiento de datos distribuido se evaluaron Apache Hadoop y Apache Spark.

Alternativas del empaquetado en contenedores

Docker es una plataforma de contenedores de código que “crea herramientas simples y un enfoque de empaquetado universal que agrupa todas las dependencias de la aplicación dentro de un contenedor que luego se ejecuta en Docker Engine”[1]. También cuenta con una amplia gama de herramientas y complementos disponibles; entre ellas está Docker Compose, Docker Swarm, Docker Hub y Docker Machine. Docker también es compatible con una gran variedad de sistemas operativos y lenguajes de programación, lo que lo vuelve una herramienta multifuncional.

Kubernetes es una herramienta de orquestación de contenedores ampliamente utilizada que ofrece una gran cantidad de características y funcionalidades avanzadas, como el escalado automático y la programación de trabajos.
Sin embargo, puede ser complejo el aprendizaje de esta aplicación, la configuración y administración pueden requerir más experiencia técnica que las otras opciones.

ALTERNATIVAS DEL DESPLIEGUE EN UN CLUSTER DE PROCESAMIENTO DE DATOS DISTRIBUIDO

Apache Spark es un sistema de procesamiento de datos de código abierto que ofrece un alto rendimiento y capacidades de procesamiento distribuido, en otras palabras, un cluster de procesamiento de datos distribuido. Además, Spark ofrece una variedad de bibliotecas y herramientas para el procesamiento y análisis de datos, como Spark SQL, Spark Streaming, etc.

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. Además, Hadoop ofrece un modelo de programación llamado MapReduce para el procesamiento paralelo de datos en lotes. Hadoop también cuenta con otras herramientas como Hadoop YARN para la gestión de recursos y Apache Hive para consultas SQL en grandes conjuntos de datos.

Después de considerar cuidadosamente cada una de estas alternativas, hemos seleccionado Docker como la mejor opción para el empaquetado de nuestra aplicación en contenedores y para el procesamiento de datos distribuido Apache Spark según nuestras necesidades, especialmente para la realización de microservicios. Además de la facilidad de uso de ambas, y la amplia gama de herramientas que se presentan, a su vez ya hemos venido aprendiendo a como hacer el uso de ellas gracias a las prácticas realizadas en clase el cual han aportado de gran manera a los conocimientos desarrollados. Por lo tanto, confiamos en que esta selección se adaptará de una forma más eficiente a las necesidades de nuestra aplicación.

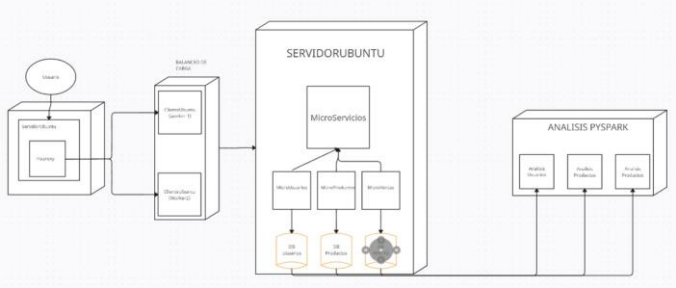
PROPUESTA DE PIPELINE DE LOS COMPONENTES
Fig.1:” Pipeline de los componentes”

Lo primero es obtener el dataset desde kaggle, posteriormente se va a hacer uso de la herramienta de apache spark para hacer un analisis de datos masivos debido a la cantidad de datos.
Debido a que pyspark esta inpirado en python podemos utilizar librerias del mismo fuente de codigo, numpy y

matplotlib para la graficación y el analisis del mismo en pyspark. El resultado final se guarda en formato de imagen y se le muestra al cliente final en una pagina web realizada en apache.

DIAGRAMA DE LOS COMPONENTES A UTILIZAR

Fig. 2: “Diagrama de componentes”



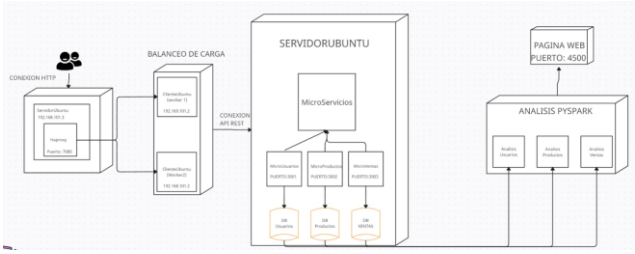
El usuario se va a conectar a una maquina del servidorUbuntu la cual cuenta con la ip 192.168.100.2 con un haproxy que se encuentra en el puerto 7080 que distribuye la carga al clienteUbuntu.
Cada worker va a estar trabajando con una conexión de API rest al servidorUbuntu con la misma ip.
Van a haber 3 micro servicios funcionando, microUsuarios en el puerto 3001, microProductos en el 3002 y microVentas enel puerto 3003. Cada uno va estar conectado a su respectiva base datos, esta base de datos estaran directamente conectadas para el analisis de pyspark.

Relación y flujo de trabajo entre los componentes

En un inicio entre el usuario al servidorUbuntu que internamente va estar con un balanceo de carga con haproxy, ese balanceo de carga vaser desde nuestra maquina clienteUbuntu donde crean 2 workers.
Internamente en el servidorUbuntu se encuentran los micro servicios funcionando, los cuales se encuentran en un contenedor docker.
El analisis de pyspark trabaja directamente con la base de datos para asi evitar estar haciendo constantemente solicitudes a los micro servicios, y asi asegurar que cada vez que se haga un cambio efectivamente se vea reflejado en el analisis.

DIAGRAMA DE DESPLIEGUE

Fig.3: “Diagrama de despliegue”



IMPLEMENTACIÓNY PUREBAS

La solución implementada consta de una aplicación con diferentes funciones. Se utilizó Docker para el despliegue de la aplicación y Apache Spark para el análisis distribuido del dataset.
Los componentes necesarios para la aplicación se organizaron en una carpeta, donde se despliegan como servicios. Se crearon imágenes de Docker para la mayoría de los servicios, incluyendo microProductos, microUsuarios, microVentas, haproxy y microWeb.
Se utilizó un archivo docker-compose.yml para llamar a todos los servicios necesarios y desplegar la aplicación.

PRUEBA DE FUNCIONAMIENTO

La aplicación implementada se puede probar ingresando la dirección IP del servidor seguida del puerto 5080 en un navegador (http://192.168.100.2:7080/). Al acceder a la dirección se muestra un inicio de sesión en la que se deben ingresar las credenciales, dependiendo del tipo de usuario (Vendedor o cliente) se muestra una vista diferente.
Si el usuario es vendedor, se muestra un menú en el que puede ver los usuarios, los productos y las ordenes de venta que se han realizado por los clientes. En el apartado usuarios tiene acceso a la información de todos los usuarios (Nombre, correo, usuario y contraseña), además de tener la función de editarlos, eliminarlos y crear nuevos usuarios.
Perspectiva vendedor: https://youtu.be/GOqbCVAEXX0
Si el usuario es cliente se le mostrará una página en la que podrá ver los productos disponibles, la cantidad y una opción para seleccionar la cantidad a comprar.
Perspectiva cliente: https://youtu.be/KLn6hAYpZgc

CONCLUSIONES

Del trabajo previo, se deduce que la implementación de una arquitectura basada en microservicios con integración a una API REST facilita la división de una aplicación en módulos independientes y autónomos, proporcionando una mayor flexibilidad en el proceso de desarrollo de software.

Esta estrategia mejora la eficiencia en la gestión de recursos, ofrece una escalabilidad superior y permite adaptaciones rápidas ante cambios en los requerimientos. Como resultado, se logra un desarrollo más dinámico y una capacidad de respuesta más efectiva en el contexto empresarial moderno.

Lo descrito anteriormente conduce a la adopción de un balanceador de carga, una herramienta esencial para evitar la sobrecarga de una aplicación o sitio web con demasiadas solicitudes a un solo servidor, facilitando así la distribución equitativa de la carga de trabajo.

La integración de Apache Spark en el análisis distribuido del proyecto permite el procesamiento eficaz de grandes volúmenes de datos, resultando en la creación de estadísticas detalladas y significativas.

El uso de Docker para el empaquetado de un clúster ha mejorado significativamente la escalabilidad y la orquestación de los contenedores que alojan los microservicios. Esta estrategia ha facilitado una gestión de recursos más eficiente y ha proporcionado una respuesta rápida y adaptable a las necesidades en constante evolución de la aplicación.

Una arquitectura bien diseñada es crucial para asegurar un sistema de alto rendimiento y prevenir dificultades como los cuellos de botella. Resulta esencial tener en cuenta factores como la escalabilidad, la eficiencia en el procesamiento y la optimización de recursos a lo largo del proceso de diseño e implementación del sistema.

REFERENCIAS

[1] (2013) Docker Website. [Online]. Disponible en: https://www.docker.com/products/container-runtime/