Proyecto de curso Infraestructura

Isabella Pérez C, Luz Angela Carabali y Nicolas Zapata

Universidad Autónoma de Occidente

Cali, Colombia.

Selección del dataset objetivo

En primer lugar, para la aplicación se necesitó una base de datos que incluyera información de la fecha, el nombre del plato, precio del plato, cantidad, frecuencia de pedido, tiempo de preparación en minutos y las valoraciones. Para ello, se hizo uso de data work.com, una plataforma web que contiene herramientas y recursos para la ciencia de datos.

El dataset objetivo, llamado “Dish”, proporciona información sobre los platos de un restaurante en España durante el año 2023, desde enero hasta diciembre. Este dataset incluye detalles sobre los platos como, por ejemplo, el tiempo de preparación que toma cada plato según el día, y la valoración, la cual define cómo fue evaluado ese plato en una escala del 1 al 10, donde 1 es mal y 10 es bueno, con el fin de mejorar la calidad del restaurante. Este dataset cuenta con un número de 7 columnas en total, y 2582 filas. En él se proporciona información detallada sobre la cantidad disponible de cada plato, la frecuencia de pedido del plato, cantidad disponible, fecha, nombre del platillo, tiempo que tarda en crearse el platillo y la valoración de este.

Criterios de Selección: Es esencial que el dataset incluya al menos el nombre y el precio de cada plato para ser considerado válido.

Validación y Limpieza: Eliminación de duplicados: Se identifican y eliminan los registros duplicados para asegurar la integridad de los datos.

Corrección de errores: Se corrigen errores tipográficos y valores fuera de rango, y se estandarizan las unidades de medida para garantizar la precisión de los datos.

Tamaño del Dataset: Amplio número de registros: Se busca que el dataset contenga un gran número de registros para proporcionar un análisis más robusto y representativo de los datos del restaurante.

Alternativas de solución:

1. **Empaquetado:**
   - **Docker**: Es una plataforma de código abierto que permite empaquetar, distribuir y ejecutar aplicaciones en contenedores. Los contenedores Docker encapsulan todo lo necesario para ejecutar una aplicación, incluidas las bibliotecas, las herramientas y el código, lo que garantiza que la aplicación se ejecute de manera uniforme en cualquier entorno.
   - **Kubernetes**: Plataforma de orquestación de contenedores de código abierto que automatiza la implementación, escalado y gestión de aplicaciones en contenedores. Permite a los equipos de desarrollo implementar aplicaciones en entornos de producción de manera más rápida y confiable, al tiempo que garantiza la disponibilidad, el escalado automático y la autorreparación de las aplicaciones. Kubernetes simplifica la administración de contenedores a escala, lo que lo hace ideal para despliegues de aplicaciones en infraestructuras de nube y entornos híbridos.
   - **AWS Fargate**: Es un servicio de contenedores completamente administrado que permite ejecutar contenedores Docker en AWS sin necesidad de aprovisionar o administrar servidores. Fargate elimina la necesidad de gestionar la infraestructura subyacente, permitiendo a los desarrolladores centrarse en la construcción y ejecución de aplicaciones.

2. **Alternativas de Procesamiento Distribuido:**
   - **Apache Hadoop**: framework de software de código abierto diseñado para el almacenamiento y procesamiento distribuido de conjuntos de datos grandes en clusters de computadoras. Hadoop se compone principalmente de dos componentes principales: el sistema de

almacenamiento distribuido Hadoop Distributed File System (HDFS) y el framework de procesamiento distribuido MapReduce. Hadoop es ideal para aplicaciones que requieren procesamiento de datos escalable y tolerante a fallos, como el procesamiento de datos en lotes.

- **Apache Hive**: herramienta de análisis de datos construida sobre Hadoop que proporciona una interfaz similar a SQL para consultar y analizar datos almacenados en Hadoop. Hive permite a los usuarios ejecutar consultas SQL complejas sobre grandes conjuntos de datos distribuidos sin necesidad de escribir código MapReduce.
- **Apache Spark**: framework de procesamiento de datos distribuido de código abierto que ofrece un rendimiento rápido y un amplio conjunto de herramientas para el procesamiento de datos en tiempo real, análisis de datos y aprendizaje automático. Spark proporciona APIs en varios lenguajes de programación, lo que lo hace accesible para una amplia gama de usuarios. Spark es conocido por su capacidad para procesar datos en memoria y su eficiencia en la ejecución de tareas complejas de análisis de datos.

Alternativas de solución seleccionadas:

1. **Docker**:

   Docker se destaca como la opción preferida en primer lugar por su enfoque en la portabilidad que garantiza que las aplicaciones empaquetadas sean altamente transportables entre diferentes entornos, ya sea en un servidor local, en la nube o en un centro de datos, lo que asegura que la aplicación funcione de manera consistente sin importar las diferencias en el entorno de ejecución.

   Además, Docker ofrece un alto nivel de aislamiento de recursos, lo que significa que cada contenedor tiene su propio entorno de ejecución independiente, incluidos el sistema de archivos, las variables de entorno y los recursos de red. Este aislamiento garantiza que las aplicaciones empaquetadas en contenedores Docker sean independientes entre sí y no interfieran unas con otras, lo que es especialmente valioso en entornos de desarrollo y producción donde múltiples aplicaciones deben coexistir en el mismo servidor.

   Otra ventaja clave es que cuenta con un ecosistema robusto de herramientas y servicios que facilitan el desarrollo, el despliegue y la gestión de aplicaciones en contenedores. Desde Docker Compose para la gestión de aplicaciones compuestas hasta Docker Swarm para la orquestación de contenedores a nivel de clúster, este ecosistema ofrece una amplia gama de herramientas que satisfacen las necesidades de desarrollo y despliegue de aplicaciones en cualquier escala.

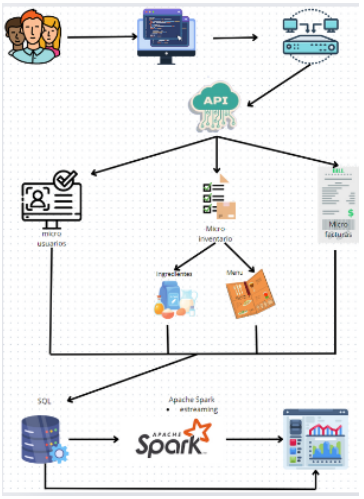
2. **Apache Spark**:

   En primer lugar, su arquitectura está diseñada para ofrecer un rendimiento excepcional al procesar grandes volúmenes de datos. Spark aprovecha la memoria de manera eficiente, lo que permite realizar operaciones en memoria, reduciendo así los tiempos de procesamiento en comparación con el acceso a disco tradicional.

   Además, Spark proporciona APIs en varios lenguajes de programación, esto hace que sea más accesible y flexible para diferentes equipos y proyectos. Además, ofrece un conjunto completo de operaciones de alto nivel que simplifican el desarrollo de aplicaciones de análisis de datos complejas.

   Otro punto fuerte de Spark es su amplia compatibilidad con diversos tipos de datos y fuentes de datos por lo que es adaptable a una variedad de necesidades y entornos, desde datos estructurados hasta datos no estructurados y semiestructurados.

Definición de la arquitectura completa del sistema:

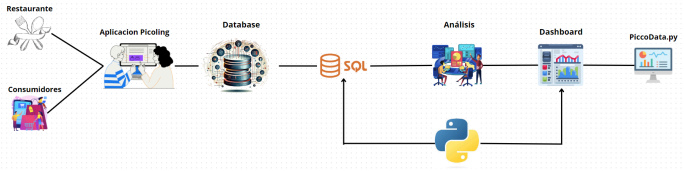


1. **Usuarios**: Representan a las personas que interactúan con el sistema. Se conectan directamente a la aplicación web.
2. **Aplicación Web**: Es la interfaz a través de la

cual los usuarios acceden al sistema. Se conecta al balanceador de carga.

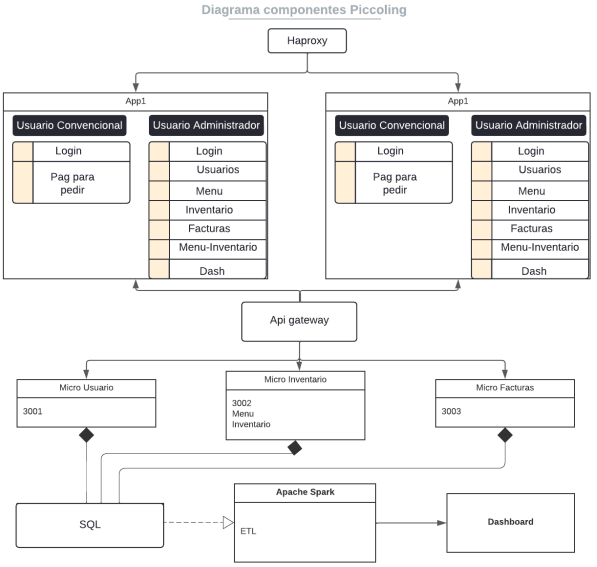
3. **Balanceador de Carga**: Distribuye las solicitudes de los usuarios entre varios servidores para mantener la estabilidad y escalabilidad del sistema.
4. **API REST**: Proporciona una interfaz para que los usuarios y la aplicación web se comuniquen con los microservicios.
5. **Microservicios**:
    - **Usuarios**: Maneja la lógica relacionada con los usuarios.
    - **Facturas**: Gestiona la información de facturación.
    - **Menú**: Se divide en dos tablas: Inventario y Preparación. Estas tablas contienen datos relacionados con los productos y su disponibilidad.
6. **Base de Datos (MYSQL)**: Almacena los datos del sistema. Las tablas de inventario y preparación se encuentran aquí.
7. **Apache Spark**: Se utiliza para el análisis de datos. Puede procesar grandes volúmenes de información y generar insights.
8. **Dashboard**: se crea un **dashboard** que presenta visualmente los resultados del análisis.

Propuesta del pipeline a utilizar:



1. **Restaurante:** Es la fuente inicial de datos. Aquí es donde se recopila la información.
2. **Consumidores:** Representan a las personas que interactúan con el sistema. Los consumidores están conectados al siguiente componente.
3. **Aplicación Piccoling:** Es la aplicación que gestiona los datos. Puede ser una aplicación web . Aquí se procesa y transforma la información.
4. **Base de Datos (SQL):** Es donde se almacenan los datos procesados. La base de datos SQL utiliza el lenguaje SQL para consultas y manipulación de datos.
5. **Análisis:** En esta etapa, los datos se analizan para extraer información relevante. Puede implicar la participación de analistas o algoritmos de procesamiento.
6. **Python (Pictodata.py):** Se utiliza un código para recopilar, procesar y analizar los datos almacenados en la aplicación web.
7. **Dashboard:** Representa la visualización de los resultados del análisis. Los gráficos y tablas proporcionan información útil para la toma de decisiones.

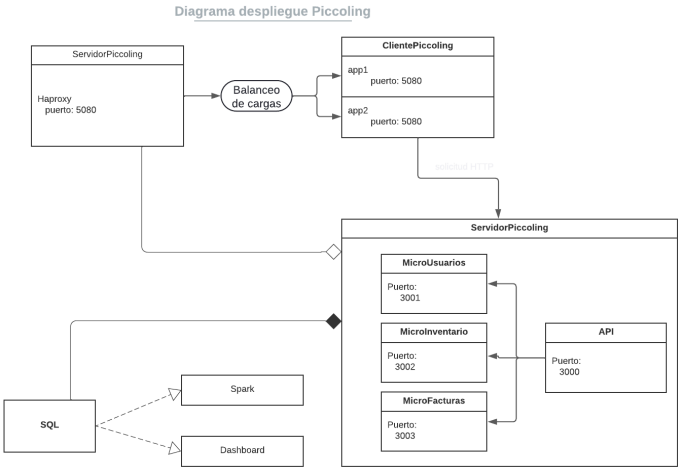
Diagrama de los componentes a utilizar:



1. **HAProxy**: Servicio que configura el balanceador de carga utilizando comandos HTTP y RESTful.
2. **Usuario Convencional y Usuario Administrador de las apps balanceadas**: Estos usuarios interactúan con la aplicación y se conectan a través de la API de Data Plane. El usuario convencional tiene acceso a la "Página para comprar", mientras que el usuario administrador tiene acceso a funcionalidades más amplias, como la gestión de usuarios, inventario, menú, facturas y la visualización de dashboard para la toma de decisiones informadas.
3. **Microservicios**:
    - **Microservicio de usuarios**: Se encarga de la gestión de los usuarios. Proporciona funcionalidades para el registro de nuevos usuarios, la autenticación de usuarios existentes y la creación de perfiles de usuario
    - **Microservicio de Inventario**: Se encarga de la gestión del inventario del restaurante. Proporciona funcionalidades para el seguimiento de los ingredientes y menús disponibles y la actualización del inventario cuando se realiza un pedido.
    - **Microservicio de Facturas:** Se encarga de la generación y gestión de las facturas. Proporciona funcionalidades para crear nuevas facturas cuando se realiza un pedido.
4. **Base de Datos (SQL)**: Almacena los datos del sistema y está conectada a los microservicios para interactuar con la base de datos.
5. **Apache Spark**: Se utiliza para el procesamiento de datos (ETL). Puede transformar, analizar y cargar datos.
6. **Dashboard**: Proporciona una interfaz visual para mostrar métricas y resultados relacionados con las facturas.
7. **Python (Pictodata.py)**: Extrae los datos

generados por la aplicación y genera el dashboard de las facturas.

Diagrama de despliegue:



1. **Servidores**:
    - **ServidorPiccoling**: Representa un servidor específico en el cual se corren los servicios de la aplicación; está conectado a un **Balanceo de cargas (haproxy)** el cual Se encarga de distribuir las solicitudes entrantes a los nodos especificados en el archivo .yml.
    - **ClientePiccoling**: Este servidor aloja la interfaz web de nuestra aplicación.
2. **Microservicios**:
    - Están conectados al ServidoPiccolcing y se ejecutan en puertos específicos:
        - **MicroUsuarios (Puerto: 3001)**: gestiona la lógica relacionada con los usuarios.
        - **MicroInventario(Puerto: 3002)**: Relacionado con el inventario de platillos e ingredientes.
        - **MicroFracturas (Puerto: 3003)**: Maneja las facturas creadas.
3. **Base de Datos (SQL)**:

    Almacena los datos del sistema para su utilización por parte de los microservicios y el análisis de datos.

4. **Apache Spark**:

    Se utiliza para el procesamiento de datos (ETL). Puede transformar, analizar y cargar.

5. **Dashboard**:

    Proporciona una interfaz visual para mostrar métricas y resultados relacionados con el sistema.

Implementación solución:

- **Problema:**

  En la industria restaurantera, la congestión durante el proceso de toma de pedidos es un desafío significativo que afecta la eficiencia y la satisfacción del cliente. Las principales causas incluyen una interfaz de pedido poco eficiente, problemas de organización en la cocina o en la atención al cliente, y una base de datos no optimizada. Estos factores generan demoras en la entrega de alimentos y pueden afectar negativamente la reputación del restaurante. Es crucial priorizar la optimización de los procesos de pedido y atención al cliente para garantizar un servicio rápido y eficiente, lo que mejora la experiencia del usuario y fortalece la posición en el mercado.

- **Solución:**

  Para abordar los problemas de congestión en el proceso de pedidos y mejorar la eficiencia operativa del restaurante, se propone el desarrollo de una aplicación web integral con las siguientes características y funcionalidades:
    - **Interfaz de usuario intuitiva**: La aplicación proporcionará una interfaz fácil de usar para usuarios y administradores, permitiendo el registro, inicio de sesión, consulta del menú y realización de pedidos en línea de manera sencilla. Los administradores podrán gestionar pedidos, actualizar el menú y acceder a la base de datos de usuarios para mejorar la toma de decisiones.
    - **Microservicio de Usuarios** seguro y escalable: Este componente gestionará las operaciones de registro, inicio de sesión y autenticación de usuarios, garantizando la seguridad y permitiendo la escalabilidad del sistema.
    - **Microservicio de Facturación**: Al generar órdenes para los administradores y actualizar automáticamente el inventario de ingredientes, este microservicio garantizará la precisión en la gestión del inventario y la facturación.
    - **Microservicio de Inventario**: Mostrará el menú y los ingredientes disponibles, y actualizará el inventario después de cada pedido, lo que evitará desabastecimientos y excesos y mejorará la planificación de compras y recursos.

  Esta solución basada en una aplicación web no solo mejorará la experiencia del usuario y la

eficiencia interna del restaurante, sino que también reducirá los tiempos de espera y aumentará la satisfacción del cliente. Con estas herramientas, el restaurante estará mejor preparado para enfrentar los desafíos de la industria y mejorar su competitividad.

- **Implementación de análisis:**
    1. **Análisis del Dataset**: En esta sección, se utilizó un conjunto de datos denominado Dish, que contiene información sobre aplicaciones y acumuladores. Este dataset se procesó mediante modelos de programación distribuida utilizando Apache Spark.
    2. **Aplicaciones en Spark:** se creó una aplicación Spark para procesar el conjunto de datos Dish. Los pasos clave en el desarrollo de la aplicación incluyeron:
    - Definición del Programa Spark: Se escribió un programa en PySpark para procesar los datos.
    - Creación de un Contexto Spark: Se inicializó un SparkContext, que es el punto de entrada para cualquier funcionalidad Spark.
    - Ejecución de la Aplicación en un Clúster: La aplicación se ejecutó en un clúster de Spark, permitiendo el procesamiento distribuido de los datos.
    3. **Acumuladores en Spark:** Para realizar un seguimiento de variables importantes durante el procesamiento de grandes volúmenes de datos del dataset Dish.

Pruebas de escalabilidad y desempeño:

Las pruebas de escalabilidad y desempeño se llevaron a cabo utilizando Apache JMeter, una herramienta de software de código abierto ampliamente reconocida en la industria para evaluar el rendimiento de aplicaciones y sistemas web. Con JMeter, pudimos simular una carga sobre nuestra aplicación, permitiéndonos medir su capacidad para manejar un alto volumen de solicitudes simultáneas y evaluar su rendimiento bajo diferentes condiciones de carga. Las pruebas abarcan diversos escenarios para garantizar una cobertura exhaustiva, y los resultados obtenidos nos proporcionaron información valiosa para identificar áreas de mejora y optimización en nuestra aplicación.

- **Resultados de las pruebas de carga normal: 100 users x 10 Loop Count**

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent
HTTP Request	1000	112	1	3689	476.41	0.00%	221.5/sec	1350.10	
TOTAL	1000	112	1	3689	476.41	0.00%	221.5/sec	1350.10	

- **Resultados de las pruebas de carga alta: 1000 users x 10 Loop Count**

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	8000	1577	1	32305	5182.94	0.27%	237.7/sec	1446.07	33.79	6230.6
TOTAL	8000	1577	1	32305	5182.94	0.27%	237.7/sec	1446.07	33.79	6230.6

- **Resultados de las pruebas de estrés: 2000 users x 10 Loop Count**

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	19360	1506	1	32384	5018.59	0.34%	146.5/sec	891.26	20.82	6228.0
TOTAL	19360	1506	1	32384	5018.59	0.34%	146.5/sec	891.26	20.82	6228.0

Conclusión

En conclusión, el proyecto “Restaurante Piccoling” presenta una solución integral y bien fundamentada para abordar los desafíos de congestión en el proceso de toma de pedidos en restaurantes. A través de una cuidadosa selección de un dataset objetivo, la implementación de una arquitectura robusta y la adopción de tecnologías avanzadas como Docker para el empaquetado de la aplicación y Apache Spark para el procesamiento distribuido, se ha logrado mejorar significativamente la eficiencia operativa y la experiencia del cliente.

La elección de un dataset rico en información relevante sobre los platos de un restaurante permitió realizar análisis detallados y generar insights valiosos para la toma de decisiones informadas. Además, se aplicaron criterios rigurosos de selección y validación de datos para garantizar la integridad y precisión de la información utilizada en el proyecto.

La arquitectura del sistema propuesta ofrece una estructura sólida y escalable, que abarca desde la interfaz de usuario hasta la gestión de datos y el análisis avanzado. Los microservicios permiten una mayor modularidad y flexibilidad en el desarrollo y mantenimiento del sistema, mientras que la integración de tecnologías como Apache Spark garantiza un procesamiento rápido y eficiente de grandes volúmenes de datos.

Las pruebas exhaustivas realizadas durante la implementación del proyecto, tanto en términos de escalabilidad como de rendimiento, han demostrado la robustez y fiabilidad de la solución propuesta. Esto proporciona una base sólida para su adopción y despliegue en entornos reales de restaurantes, donde la eficiencia operativa y la satisfacción del cliente son fundamentales para el éxito del negocio.

En última instancia, este proyecto no solo aborda los desafíos específicos de congestión en el proceso de toma de pedidos en restaurantes, sino que también sienta las bases para futuras mejoras y optimizaciones en áreas relacionadas con la gestión de datos y la experiencia del cliente. Con una implementación cuidadosa y una atención continua a la retroalimentación del usuario, esta solución tiene el potencial de transformar positivamente la forma en que los restaurantes operan y brindan servicio a sus clientes.

Referencias

1. Apache Hadoop. (s/f). Apache.org. de https://hadoop.apache.org/
2. Apache Hive. (s/f). Apache.org, de https://hive.apache.org/
3. Power BI. (s/f). Microsoft.com, de https://www.microsoft.com/es-es/power-platform/products/power-bi
4. PySpark Overview — PySpark master documentation. (s/f). Apache.org, de https://spark.apache.org/docs/latest/api/python/index.html
5. ¿Qué es Kubernetes? (s/f). Kubernetes.io, de https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/
6. Ratliff, S. (2022, mayo 10). Docker: Accelerated container application development. Docker. https://www.docker.com/
7. Welcome to. (s/f). Python.org, de https://www.python.org/
8. (S/f-a). Mysql.com., de https://www.mysql.com/
9. (S/f-b). Amazon.com. de https://aws.amazon.com/es/fargate/
10. piccoling-whit-docker. Github.com, from https://github.com/isabellaperezcav/piccoling-whit-docker