Proyecto - sale cars

Selección del dataset objetivo

I. Análisis

1. Selección del dataset objetivo

Para la implementación del sistema de análisis de datos del proyecto, se seleccionó el dataset titulado **“UAE Used Car Prices and Features - 10k Listings”**, disponible públicamente en la plataforma Kaggle (enlace al dataset). Este conjunto de datos contiene más de 10.000 registros de vehículos usados listados para venta en los Emiratos Árabes Unidos, con atributos relevantes para el análisis comercial y técnico del mercado automotor.

Entre las variables más representativas del dataset se encuentran:

- **Marca y modelo:** make, model
- **Año de fabricación:** year
- **Kilometraje:** mileage
- **Precio de venta:** price
- **Transmisión:** transmission
- **Tipo de combustible:** Fuel Type
- **Color del vehículo:** Color
- **Cilindraje y tamaño del motor:** engine_size, Cylinders

Este dataset fue transformado y analizado usando PySpark en un clúster de procesamiento distribuido. Se diseñaron múltiples consultas SQL sobre vistas temporales generadas en Spark para responder preguntas de negocio como:

- ¿Cuál es el precio promedio por marca?
- ¿Qué marcas y modelos son más populares?
- ¿Cuál es la distribución de vehículos por tipo de transmisión y tipo de combustible?
- ¿Qué colores son más comunes?
- ¿Cómo varía la edad promedio de los vehículos por marca?
- ¿Qué vehículos presentan menor kilometraje con mayor valor?

Los resultados se almacenaron en una base de datos MySQL y se visualizaron mediante Metabase.

2. Generación y selección de alternativas de solución

Empaquetado y despliegue de la aplicación

Se evaluaron las siguientes alternativas:

- **Docker Compose:** útil para desarrollo y pruebas locales, pero limitado en capacidades de orquestación en producción.
- **Docker Swarm (seleccionado):** balance entre simplicidad y funcionalidad, con soporte nativo en Docker Engine y suficiente para las necesidades del proyecto.

Se optó por **Docker Swarm** como motor de orquestación, acompañado de archivos Dockerfile por microservicio y un docker-compose.yml con extensión para despliegue en modo Swarm. Esto permitió manejar réplicas, escalabilidad y balanceo de carga de forma integrada.

Procesamiento de datos a gran escala

Para la segunda parte del proyecto, se consideraron las siguientes herramientas:

- **Hadoop:** descartado por su complejidad operativa en el entorno planteado.
- **Dask:** opción viable, pero con menor madurez que otras alternativas.
- **Apache Spark (seleccionado):** herramienta robusta, bien documentada, con soporte nativo para SQL, transformaciones sobre DataFrames y ejecución distribuida.

El procesamiento se realizó en un clúster Spark desplegado en Docker Swarm con un nodo master y múltiples workers. La lógica de transformación se codificó en el script pipeline_spark_vehiculos.py.

3. Definición de la arquitectura completa del sistema

El sistema fue dividido en dos subsistemas principales: **Aplicación Web (microservicios)** y **Sistema de análisis de datos distribuidos**.

a) Aplicación Web - Microservicios REST

Componentes:

- **Usuarios:** registro, autenticación, gestión de perfiles.

- **Vehículos:** CRUD de anuncios de autos disponibles para la venta.
- **Compras:** agendamiento de citas y simulación de procesos de compra.

Tecnologías:

- **Backend:** Node.js con Express, JWT, bcrypt, mysql2.
- **Base de datos:** MySQL 8.0 (contenedorizado).
- **Frontend:** React.js (SPA) con Tailwind CSS, React Router, Axios.

b) Procesamiento distribuido y visualización

- **Clúster Spark:** compuesto por un contenedor master y múltiples workers utilizando la imagen oficial de Bitnami Spark.
- **Contenedor PySpark Runner:** ejecuta el pipeline que:
  - Carga y transforma el dataset CSV.
  - Realiza múltiples consultas SQL.
  - Exporta los resultados a MySQL.
- **Metabase:** conectado a la base de datos, genera dashboards y reportes dinámicos con filtros. Estos dashboards se integran al frontend mediante iframes.

c) Comunicación entre componentes

Aunque no se utilizó un API Gateway ni una capa intermedia para orquestar el flujo de datos entre el sistema analítico y la aplicación web, ambos subsistemas comparten la misma base de datos MySQL como mecanismo de integración. Esto permite que los resultados de análisis generados por Spark estén disponibles directamente para consulta por Metabase y visualización embebida en la interfaz web.

II. Diseño

A. Propuesta del pipeline, componentes y algoritmos utilizados

Para el procesamiento de datos y análisis se diseñó un pipeline único, desarrollado con PySpark, que se ejecuta en un clúster distribuido de Apache Spark. El pipeline tiene las siguientes etapas técnicas:

1. **Inicialización del contexto Spark:** Se crea una sesión Spark con configuración específica para conectar con el maestro del clúster (spark://spark-master:7077) y se carga el conector JDBC para MySQL. Esto

habilita la lectura, procesamiento y escritura de datos entre Spark y la base relacional.

2. **Carga del dataset:** Se importa el archivo CSV uae_used_cars_10k.csv que contiene aproximadamente 10,000 registros con información de vehículos usados. Se utiliza el método spark.read.csv con opción de encabezados y detección automática de tipos de datos.
3. **Preprocesamiento y limpieza:** Se descartan filas que presenten valores nulos en campos críticos (make, model, price, year, mileage) para asegurar calidad y coherencia en el análisis posterior.
4. **Transformaciones y enriquecimiento:** Se agregan columnas calculadas para facilitar análisis futuros:
    a. make_upper: representación de la marca en mayúsculas para uniformidad.
    b. price_usd: normalización del precio como valor flotante.
    c. car_age: cálculo de la edad del vehículo tomando el año base 2025.
5. **Creación de vista SQL temporal:** El DataFrame resultante se registra como una vista temporal llamada vehiculos, permitiendo la ejecución de consultas SQL nativas de Spark para análisis avanzado.
6. **Persistencia en base de datos relacional:** Se implementa una función genérica para guardar resultados de consultas en tablas MySQL a través de JDBC, empleando modo de sobreescritura para mantener datos actualizados.
7. **Consultas analíticas ejecutadas:** Se definieron once consultas SQL que generan tablas con métricas clave del mercado automotor, entre ellas:
    a. Precio promedio por marca.
    b. Conteo de vehículos por año de fabricación.
    c. Distribución de vehículos según rango de precio.
    d. Modelos más vendidos por marca.
    e. Vehículos con menor kilometraje.
    f. Precio promedio segmentado por tipo de transmisión y combustible.
    g. Edad promedio de vehículos por fabricante.
    h. Precio promedio por número de cilindros.
    i. Selección de vehículos con precio superior a 50,000 y kilometraje menor a 100,000 km.
8. **Cierre de sesión Spark:** Finalmente, se cierra la sesión Spark para liberar recursos del clúster.

Este pipeline permite procesar grandes volúmenes de datos en paralelo, obtener estadísticas detalladas y mantener actualizada la base de datos relacional que sirve para la visualización posterior.

Construcción, despliegue y orquestación de contenedores

La construcción de imágenes Docker para los microservicios se realizó utilizando los archivos Dockerfile específicos localizados en los siguientes directorios del proyecto:

- Nginx: /var/www/html/SalesCarsV2/Dockerfile
- Compras: /var/www/html/SalesCarsV2/backend/compras_src/Dockerfile
- Vehículos: /var/www/html/SalesCarsV2/backend/vehiculos_src/Dockerfile
- Usuarios: /var/www/html/SalesCarsV2/backend/usuarios_src/Dockerfile
- Frontend: /var/www/html/SalesCarsV2/FrontEnd/Dockerfile
- PySpark: /var/www/html/SalesCarsV2/pyspark/Dockerfile
- Metabase: /var/www/html/SalesCarsV2/metabase_arm/Dockerfile

Para los servicios de HAProxy y MySQL se utilizaron imágenes oficiales, sin construir Dockerfiles personalizados.

El proceso para construir, etiquetar y subir las imágenes al repositorio remoto siguió el flujo:

docker build .

docker image ls

docker tag mariavalencia30/:latest

docker push mariavalencia30/:latest

Por ejemplo, para el servicio nginx:

docker tag <image_id> mariavalencia30/salecars:nginx-latest

docker push mariavalencia30/salecars:nginx-latest

El despliegue del entorno de producción se realizó utilizando Docker Swarm con la siguiente metodología:

- Inicialización del clúster en el nodo manager con:

docker swarm init --advertise-addr <IP_manager>

- Incorporación de nodos worker mediante el token generado:

docker swarm join --token :2377

docker node ls

- Despliegue del stack y servicios con:

docker stack deploy -c docker-compose.yml salescarsv2

Para escalar los servicios críticos y mejorar la disponibilidad y rendimiento, se ejecutó:

docker service scale salescarsv2_vehiculos=3

Para verificar las tareas (contenedores) activas y en qué nodo se ejecutan, se usó:

docker service ps <nombre_servicio>

Se aseguró que las imágenes utilizadas en el despliegue coincidan con las construidas y etiquetadas para mantener coherencia en el ambiente.

La red de comunicación interna se configuró mediante una red overlay gestionada automáticamente por Docker Swarm, facilitando la comunicación segura y eficiente entre contenedores en distintos nodos.

B. Diagrama del pipeline

Se incluye el diagrama gráfico que ilustra la secuencia de operaciones en el pipeline, mostrando la conexión entre el dataset, las etapas de procesamiento Spark, las consultas SQL y la persistencia en MySQL.

C. Arquitectura distribuida y despliegue de microservicios en Docker Swarm

El sistema de backend y análisis se despliega en un clúster Docker Swarm formado por al menos dos nodos: un nodo manager (servidorUbuntu) y un nodo worker (clienteUbuntu).

El despliegue contiene múltiples servicios contenedorizados:

- Microservicios backend: usuarios, vehiculos y compras, cada uno con réplicas configuradas.
- Base de datos MySQL, contenedorizada y accesible para todos los servicios.
- Frontend React, orquestado y accesible a través de un balanceador de carga HAProxy.
- Clúster Apache Spark distribuido en contenedores master y worker.
- Contenedor dedicado para ejecutar el pipeline PySpark.
- Servicio Metabase para visualización de datos.

Docker Swarm utiliza un algoritmo de planificación para distribuir automáticamente las réplicas de cada servicio entre los nodos activos, balanceando carga y asegurando alta disponibilidad. Por ejemplo, al aumentar el número de réplicas de Spark workers, Swarm asigna contenedores a ambos nodos según capacidad y disponibilidad.

D. Relación y flujo de trabajo entre componentes del sistema

El flujo de datos y control en el sistema es el siguiente:

1. El dataset CSV es almacenado en un volumen compartido accesible por el contenedor PySpark.
2. El pipeline Spark carga y procesa el dataset en paralelo dentro del clúster, ejecutando transformaciones y consultas SQL.
3. Los resultados de las consultas se escriben en tablas MySQL accesibles desde cualquier nodo.
4. Metabase se conecta a MySQL y genera dashboards con gráficos y filtros dinámicos basados en los datos procesados.
5. El frontend React incrusta estos dashboards a través de iframes, facilitando a los usuarios la exploración e interacción con los reportes generados.

Este diseño permite desacoplar el procesamiento de datos del sistema de presentación, facilitando escalabilidad, mantenimiento y actualización independiente de cada módulo.

E. Diagrama de despliegue

Se incluirá el diagrama detallado que muestra los nodos físicos o virtuales del clúster, servicios desplegados en cada nodo, conexiones de red, y flujos de datos entre Spark, MySQL, Metabase y la aplicación frontend.

Sí, está bien estructurado, claro y completo para la sección "III. Implementación y Pruebas". Algunos ajustes menores para mejorar la claridad y uniformidad:

III. Implementación y Pruebas

A. Implementación de la solución

- La solución se implementó con tecnologías seleccionadas en fases anteriores.
- Microservicios backend independientes para usuarios, vehículos y compras, desarrollados en Node.js con Express y contenerizados mediante Docker.
- Frontend SPA construido en React.js, desplegado en contenedor y expuesto mediante HAProxy para balanceo de carga y proxy inverso.
- Base de datos MySQL 8.0 en contenedor con persistencia de datos, accesible para todos los servicios.
- Clúster Apache Spark desplegado con nodos master y workers en contenedores para procesamiento y análisis de datos con PySpark.
- Visualización mediante Metabase, integrado en frontend vía iframes.

B. Pruebas de funcionamiento

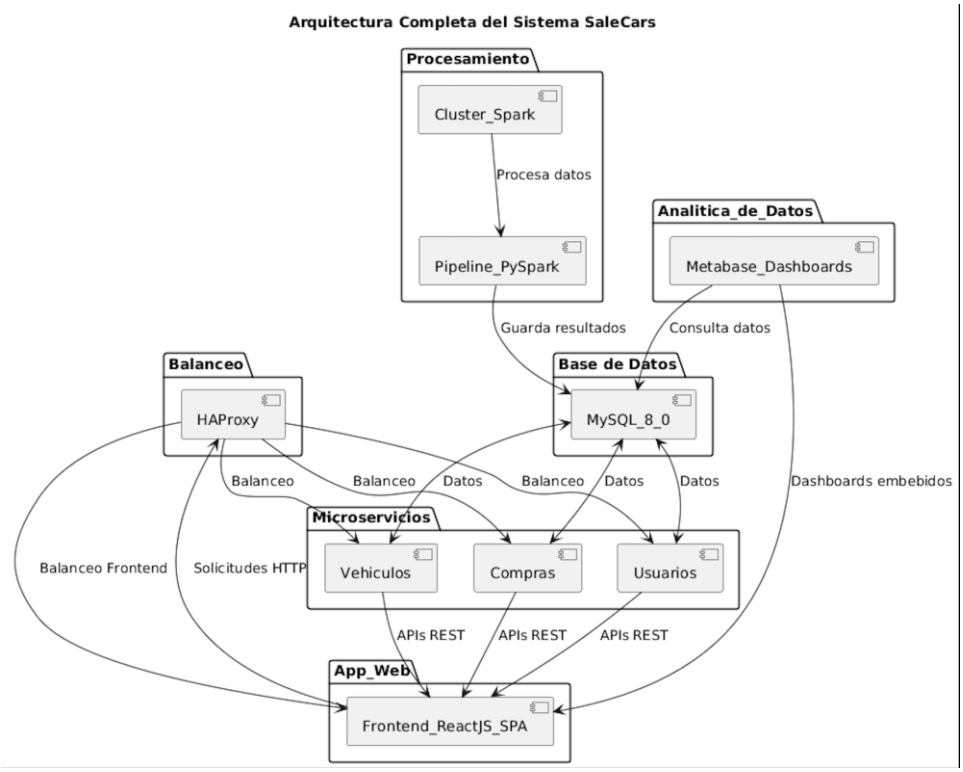
- Pruebas de carga realizadas con Apache JMeter sobre APIs de usuarios, vehículos, compras y frontend.
- Parámetros: número de hilos, periodo de subida y contador de bucle configurados para simular cargas progresivas.
- Resultados: todas las solicitudes con código HTTP 200 OK, latencias promedio entre 2-9 ms, tiempos de carga entre 2-9 segundos, y sin errores.
- Verificación de conectividad entre servicios y base de datos mediante logs y herramientas de monitoreo.
- Validación de visualización y actualización de dashboards en Metabase integrados al frontend.

C. Pruebas de escalabilidad y desempeño

- Escalado de servicios críticos (usuarios, vehículos, compras, frontend) a 2 réplicas con Docker Swarm.
- Balanceo de carga automático entre nodos manager y worker, garantizando alta disponibilidad y tolerancia a fallos.
- Pipeline PySpark ejecutándose correctamente sin necesidad de re-ejecución, con resultados almacenados en MySQL sin errores.
- Pruebas de carga en escenarios con hasta 250 hilos concurrentes y periodos de subida crecientes.

- Latencias y tiempos de carga aumentan moderadamente conforme crece la carga, manteniendo respuestas HTTP 200 OK y sin errores.

  Arquitectura



D. Contenedores desplegados

Servicio	Imagen	Réplicas	Puertos expuestos
salescarsv2_usuarios	marivalencia/salescarsv2_usuarios	2	3001/tcp
salescarsv2_vehiculos	mariavalencia30/salecars:vehiculos_tag	2	3002/tcp
salescarsv2_compras	mariavalencia30/salescarsv2-compras	2	3006/tcp
salescarsv2_frontend	mariavalencia30/salescarsv2-frontend	2	80/tcp (host 8082)
salescarsv2_db	mysql:8.0	1	3306/tcp (host 32000)

salescarsv2_haproxy	haproxy:2.8	1	80/tcp, 8404/tcp
salescarsv2_spark-master	bitnami/spark:latest	1	7077/tcp, 8081/tcp
salescarsv2_spark-worker-1	bitnami/spark:latest	1	-
salescarsv2_pyspark-runner	mariavalencia30/salecars-pyspark	1	-
salescarsv2_metabase	mariavalencia30/metabase_arm:latest	1	3008/tcp

E. Resultados por escenarios de prueba

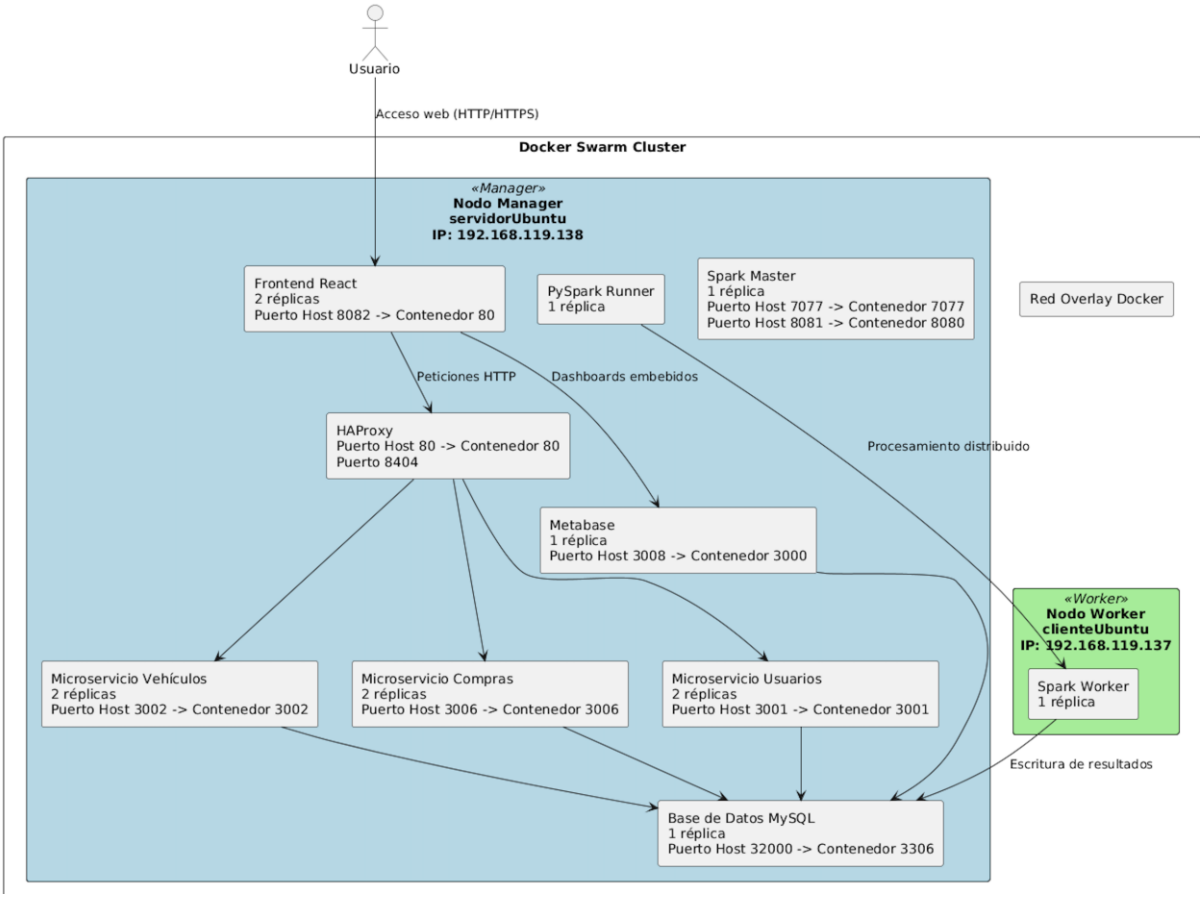
Escenario	Hilos	Periodo Subida (s)	Latencia Promedio (ms)	Tiempo Carga (s)	Código HTTP	Errores	Conclusión
1	10	10	2-3	2-3	200 OK	0	Sistema estable bajo carga baja, sin errores ni retrasos.
2	50	20	3-9	3-9	200 OK	0	Respuesta estable y adecuada bajo carga media.
3	100	30	2-3	2-3	200 OK	0	Eficiente respuesta con baja latencia a carga incrementada.
4	150	40	2-8	2-8	200 OK	0	Estabilidad mantenida con alta concurrencia.
5	200	50	1-7	1-7	200 OK	0	Respuesta eficaz a alta concurrencia sin fallos.

6	250	60	3-17	3-17	200 OK	0	Sistema responde correctamente con ligera subida en latencia.

Presentacion:

https://www.canva.com/design/DAGGGIVWSCk/uCTsC85n8JMIF3Vszm-aCQ/edit

Diagrama de Despliegue


Usuario
Acceso web (HTTP/HTTPS)
Docker Swarm Cluster
«Manager»
Nodo Manager
servidorUbuntu
IP: 192.168.119.138
Frontend React
2 réplicas
Puerto Host 8082 -> Contenedor 80
PySpark Runner
1 réplica
Spark Master
1 réplica
Puerto Host 7077 -> Contenedor 7077
Puerto Host 8081 -> Contenedor 8080
Red Overlay Docker
Peticiones HTTP
Dashboards embebidos
HAProxy
Puerto Host 80 -> Contenedor 80
Puerto 8404
Procesamiento distribuido
Metabase
1 réplica
Puerto Host 3008 -> Contenedor 3000
«Worker»
Nodo Worker
clienteUbuntu
IP: 192.168.119.137
Spark Worker
1 réplica
Microservicio Vehículos
2 réplicas
Puerto Host 3002 -> Contenedor 3002
Microservicio Compras
2 réplicas
Puerto Host 3006 -> Contenedor 3006
Microservicio Usuarios
2 réplicas
Puerto Host 3001 -> Contenedor 3001
Escritura de resultados
Base de Datos MySQL
1 réplica
Puerto Host 32000 -> Contenedor 3306