Data Lakehouse para la distribución farmacéutica

Trabajo Fin de Máster

Máster en Big Data & Data Engineering

Autor: Eric Barba Lopez

Tutores: Jorge Centeno y Alberto González

Febrero 2026

1. Resumen 4
2. Palabras clave 4
3. Introducción 5
3.1. Contextualización del proyecto 5
3.2. Objetivos del proyecto 5
3.3. Justificación e interés 6
4. Metodología 8
4.1. Diseño general del proyecto 8
4.2. Proceso de trabajo y fases (planificación) 9
5. Arquitectura 10
5.1. Diagrama general 10
5.2. Descripción de la arquitectura técnica 10
5.3. Justificación de elección de tecnologías (bases de datos, herramientas de procesamiento, orquestación, etc.) 11
5.4. Estimación básica de costes de infraestructura 13
5.5. Estrategia DevOps 13
6. Solución tecnológica 15
6.1. Fuentes de datos 15
6.2. Preparación de datos 15
6.3. Flujos de datos 16
6.4. Procesamiento (batch, streaming, etc.) 16
6.5. Orquestación 17
6.6. Almacenamiento y consulta 17
6.7. Explotación de resultados (reporting, modelos…) 18
6.8. Consideraciones éticas / legales del uso de datos (solo si aplica) 19
7. Resultados y conclusiones 20
7.1. Logros alcanzados y validación de los resultados 20
7.2. Métricas utilizadas 21
7.3. Limitaciones identificadas 22

7.4. Futuras líneas de mejora y/o desarrollo 23
8. Bibliografia 24
9. Anexos 25

1. Resumen

El presente trabajo desarrolla una plataforma de datos integral para el sector farmacéutico, implementando una arquitectura lakehouse que permite gestionar tanto el procesamiento por lotes de datos históricos como la ingesta en tiempo real de pedidos comerciales.

El proyecto aborda un problema habitual en empresas del sector: la fragmentación de datos comerciales y la dificultad para obtener una visión unificada del negocio que permita tomar decisiones basadas en información actualizada. La solución propuesta integra múltiples fuentes de datos en un repositorio centralizado siguiendo el patrón medallion (Bronze, Silver, Gold), que garantiza la trazabilidad y calidad de los datos en cada etapa del procesamiento.

La arquitectura técnica se sustenta en Apache Spark como motor de procesamiento, Delta Lake como formato de almacenamiento con capacidades ACID, Apache Kafka para la gestión de eventos en tiempo real, y Apache Airflow para la orquestación de los flujos de trabajo. Adicionalmente, se incorpora dbt para la capa de transformaciones analíticas y Spark MLlib para dos componentes de aprendizaje automático: segmentación de clientes mediante clustering y predicción de demanda.

Todo el sistema se despliega mediante contenedores Docker, lo que garantiza la reproducibilidad del entorno y facilita tanto el desarrollo como una eventual migración a infraestructura cloud. El proyecto procesa un volumen de aproximadamente 250.000 registros de ventas, segmenta 751 clientes en tres perfiles diferenciados y genera predicciones de demanda para 241 productos distintos.

Los resultados obtenidos demuestran la viabilidad de implementar arquitecturas de datos modernas en entornos locales sin dependencia de servicios cloud, manteniendo los estándares de calidad y las buenas prácticas que se aplicarían en un entorno empresarial.

2. Palabras clave

Lakehouse, Delta Lake, Apache Spark, Apache Kafka, Streaming, ETL, Medallion Architecture, Data Engineering, Machine Learning, dbt, Apache Airflow, Docker, Data Quality, Star Schema.

3. Introducción

3.1. Contextualización del proyecto

El sector farmacéutico genera diariamente grandes volúmenes de datos provenientes de múltiples fuentes: transacciones de venta, pedidos de distribuidores, información de productos, datos geográficos de puntos de venta y métricas de rendimiento comercial. Esta información, cuando se gestiona de forma fragmentada, dificulta la obtención de una visión integral del negocio y ralentiza la toma de decisiones estratégicas.

Las arquitecturas tradicionales de almacenamiento de datos, basadas en data warehouses relacionales, presentan limitaciones cuando se requiere combinar procesamiento por lotes con análisis en tiempo real. Por otro lado, los data lakes, aunque ofrecen flexibilidad para almacenar datos en su formato nativo, suelen carecer de las garantías transaccionales necesarias para entornos analíticos exigentes.

En este contexto surge el concepto de lakehouse, una arquitectura que combina las ventajas de ambos paradigmas: la flexibilidad y escalabilidad de los data lakes con las capacidades de gestión transaccional y rendimiento analítico de los data warehouses. Esta aproximación se ha consolidado en los últimos años como una alternativa viable para organizaciones que necesitan unificar sus cargas de trabajo de datos.

El presente proyecto se enmarca en el ámbito de la distribución farmacéutica, un sector caracterizado por redes comerciales extensas, catálogos de productos amplios y la necesidad de responder con agilidad a las variaciones de la demanda. La capacidad de analizar el comportamiento de compra de los clientes, predecir tendencias y evaluar el rendimiento de los equipos comerciales resulta determinante para la competitividad de las empresas del sector.

3.2. Objetivos del proyecto

El objetivo principal de este trabajo es diseñar e implementar una plataforma de datos completa que permita a una empresa farmacéutica ficticia centralizar, procesar y explotar su información comercial de manera eficiente.

Este objetivo general se descompone en los siguientes objetivos específicos:

i. Construir una arquitectura lakehouse funcional que implemente el patrón medallion con tres capas diferenciadas (Bronze, Silver, Gold), garantizando la trazabilidad de los datos desde su origen hasta su explotación final.
ii. Implementar capacidades de procesamiento híbrido que permitan tanto la ingesta por lotes de datos históricos como el procesamiento en tiempo real de eventos mediante streaming.

iii. Desarrollar un modelo dimensional optimizado para consultas analíticas, siguiendo las buenas prácticas de modelado de datos con un esquema en estrella que facilite la generación de informes y cuadros de mando.
iv. Incorporar componentes de aprendizaje automático que aporten valor analítico al negocio, concretamente un modelo de segmentación de clientes y un modelo de predicción de demanda.
v. Establecer mecanismos de calidad de datos y testing automatizado que garanticen la fiabilidad de la información procesada.
vi. Orquestar todos los flujos de trabajo mediante un sistema que permita la ejecución programada, el monitoreo y la gestión de dependencias entre tareas.
vii. Séptimo, garantizar la reproducibilidad completa del entorno mediante contenedorización, de forma que cualquier persona pueda desplegar el sistema sin dependencias de servicios externos.

3.3. Justificación e interés

La realización de este proyecto se justifica desde varias perspectivas complementarias.

Desde el punto de vista académico, el trabajo permite aplicar de forma integrada los conocimientos adquiridos a lo largo del máster en materias como arquitectura de datos, diseño de pipelines, procesamiento distribuido, bases de datos, contenedores y aprendizaje automático. No se trata de ejercicios aislados sobre cada tecnología, sino de un sistema completo donde todas las piezas deben funcionar de manera coordinada.

Desde el punto de vista técnico, el proyecto aborda retos reales de la ingeniería de datos moderna: la integración de fuentes heterogéneas, la convivencia de procesamiento batch y streaming, la gestión de la calidad del dato, el modelado dimensional y la incorporación de capacidades analíticas avanzadas. Estos son precisamente los desafíos que un ingeniero de datos encuentra en su práctica profesional.

Desde el punto de vista de negocio, la solución propuesta responde a necesidades concretas del sector farmacéutico. La segmentación de clientes permite diseñar estrategias comerciales diferenciadas según el perfil de cada punto de venta. La predicción de demanda facilita la planificación de inventarios y la asignación de recursos comerciales. Los indicadores de rendimiento permiten evaluar la efectividad de los equipos de ventas por territorio y producto.

La propuesta inicial de este proyecto contemplaba una implementación en la nube utilizando Azure como plataforma principal. El diseño original se apoyaba en Azure Databricks como motor de procesamiento, Azure Data Lake Storage Gen2 para el almacenamiento, Unity Catalog para la gobernanza de datos y Azure Monitor para la observabilidad del sistema. Sin embargo, durante la

fase de desarrollo se produjo el agotamiento del crédito académico de Azure, lo que obligó a replantear la estrategia de infraestructura.

Ante esta situación, se decidió migrar el proyecto a un entorno completamente local basado en contenedores Docker. Esta decisión, lejos de suponer una reducción del alcance técnico, aporto dos ventajas significativas. En primer lugar, elimino la dependencia de cuentas y créditos cloud, garantizando que el proyecto pudiera completarse sin restricciones económicas. En segundo lugar, y quizá más relevante desde el punto de vista formativo, obligo a comprender en profundidad el funcionamiento interno de cada componente de la arquitectura, sin las abstracciones que los servicios gestionados interponen entre el desarrollador y la tecnología subyacente.

Es importante señalar que las competencias técnicas demostradas son equivalentes en ambos enfoques. El stack final (Apache Spark, Delta Lake, Apache Kafka, Apache Airflow y dbt) es conceptualmente idéntico al propuesto originalmente y totalmente portable a un entorno cloud. De hecho, la solución adoptada incorpora herramientas como Airflow y dbt que no estaban en el planteamiento inicial y que son estándar en la industria de data engineering. La siguiente tabla resume la correspondencia entre los componentes originales y los finales:

Aspecto	Propuesta original (Azure)	Solucion final (Docker)
Procesamiento	Azure Databricks	Apache Spark 3.5.1
Almacenamiento	ADLS Gen2 + Delta Lake	Delta Lake sobre ficheros locales
Streaming	Azure Event Hubs / Kafka	Apache Kafka 4.1.1
Orquestación	Databricks Jobs	Apache Airflow 2.10.4
Transformaciones	Notebooks Databricks	dbt Core 1.11.2 + dbt-spark
Gobernanza	Unity Catalog + Key Vault	No aplica (entorno local)
Observabilidad	Azure Monitor	Logs de contenedores Docker
Infraestructura	Servicios gestionados Azure	Docker Compose (IaC)

4. Metodología

4.1. Diseño general del proyecto

El proyecto se ha abordado siguiendo una metodología iterativa e incremental, priorizando en cada fase la obtención de un sistema funcional que pudiera ampliarse progresivamente con nuevas capacidades.

El diseño parte de una premisa fundamental: construir un pipeline de datos completo y operativo antes de añadir sofisticación. Esto implica que la primera iteración debía conseguir que los datos fluyeran desde las fuentes hasta las tablas finales, aunque fuera con transformaciones básicas. Las iteraciones posteriores fueron refinando cada capa, añadiendo validaciones, optimizando el modelo dimensional e incorporando los componentes de aprendizaje automático.

La arquitectura se organizó en torno al patrón medallion, ampliamente adoptado en implementaciones lakehouse. Este patrón establece tres capas de datos con responsabilidades diferenciadas:

La capa Bronze almacena los datos en su estado original o con transformaciones mínimas. Su función es preservar la información tal como llega de las fuentes, actuando como zona de aterrizaje y permitiendo reprocesar los datos si fuera necesario.

La capa Silver contiene los datos una vez aplicadas las transformaciones de limpieza, normalización y validación. En esta capa se resuelven problemas de calidad, se unifican formatos y se eliminan duplicados. Los datos de Silver están listos para ser consumidos por procesos analíticos.

La capa Gold alberga los datos modelados para su explotación final. En el caso de este proyecto, incluye un modelo dimensional en estrella con tablas de hechos y dimensiones, así como las tablas generadas por los modelos de aprendizaje automático.

El procesamiento de datos se implementó con Apache Spark, que permite manejar tanto cargas batch como streaming con el mismo framework. Para el almacenamiento se eligió Delta Lake, un formato abierto que añade capacidades ACID sobre ficheros Parquet y que se ha convertido en un estándar de facto para arquitecturas lakehouse.

La orquestación de los flujos de trabajo se delegó en Apache Airflow, que permite definir las dependencias entre tareas, programar ejecuciones y monitorizar el estado del pipeline. Para la capa de transformaciones analíticas se incorporó dbt, una herramienta que permite definir modelos SQL con control de versiones y testing integrado.

4.2. Proceso de trabajo y fases (planificación)

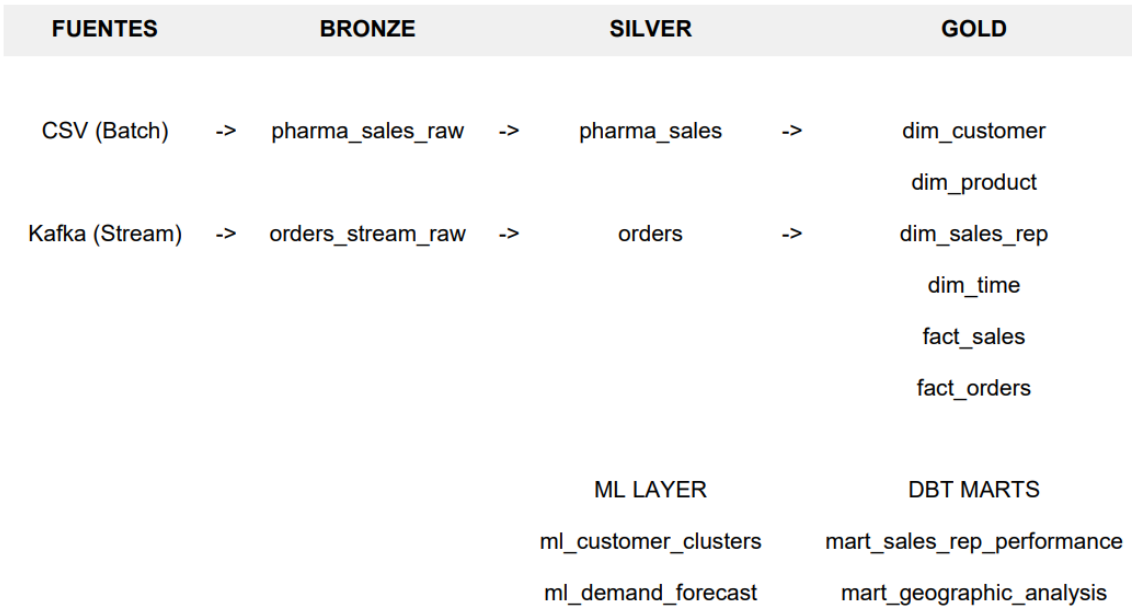
El desarrollo del proyecto se estructuró en fases secuenciales, aunque con cierto solapamiento entre ellas para mantener la agilidad.

1. Definición del alcance y la selección de tecnologías. Se evaluaron diferentes datasets públicos hasta identificar uno con la riqueza suficiente para soportar un modelo dimensional interesante. Se definió la arquitectura objetivo y se validó la viabilidad técnica de cada componente. Esta fase incluyó también la configuración inicial del entorno Docker con los servicios básicos.
2. Implementación de la capa Bronze. Se desarrollaron los scripts de ingesta para cargar el dataset de ventas farmacéuticas en formato Delta Lake. Paralelamente, se configuró Apache Kafka y se implementó un simulador de pedidos en Python que genera eventos sintéticos para alimentar el flujo de streaming. Se desarrolló el consumidor de Spark Structured Streaming que persiste estos eventos en la capa Bronze.
3. Capa Silver. Se implementaron las transformaciones de limpieza y normalización de datos, incluyendo la conversión de tipos, la estandarización de valores categóricos y la validación de reglas de calidad. Se establecieron los criterios de deduplicación y se desarrolló la lógica para gestionar registros que no cumplen las validaciones.
4. Capa Gold con el modelo dimensional. Se diseñó el esquema en estrella identificando las dimensiones relevantes (cliente, producto, representante comercial, tiempo) y las tablas de hechos correspondientes. Se implementó la generación de claves subrogadas y las relaciones entre tablas.
5. Incorporación de los componentes de aprendizaje automático. Se desarrolló el modelo de clustering para segmentación de clientes utilizando el algoritmo K-Means, incluyendo la selección del número óptimo de clusters y la interpretación de los segmentos resultantes. Posteriormente se implementó el modelo de predicción de demanda con Gradient Boosted Trees, diseñando las features temporales necesarias y evaluando la precisión de las predicciones.
6. Integración de dbt para la capa analítica. Se definieron los modelos SQL que generan los marts de análisis, se configuraron los tests de calidad de datos y se documentaron las transformaciones. Esta fase también incluyó la integración de dbt con el pipeline de Airflow.
7. Orquestación completa con Airflow. Se desarrolló el DAG que coordina todas las tareas del pipeline, estableciendo las dependencias correctas y configurando los reintentos en caso de fallo. Se verificó la ejecución completa del flujo de principio a fin.
8. Documentación, pruebas de integración y preparación de los entregables del proyecto.

5. Arquitectura

5.1. Diagrama general

La arquitectura del sistema sigue el patron medallion con tres capas de datos, orquestadas por Apache Airflow y desplegadas en contenedores Docker. El siguiente diagrama muestra los componentes principales y el flujo de datos entre ellos:



Nota: El diagrama completo en formato SVG se incluye en el repositorio del proyecto (docs/arquitectura_datos.svg)

5.2. Descripción de la arquitectura técnica

La arquitectura se compone de varios subsistemas interconectados que colaboran para procesar los datos desde su origen hasta su explotación final.

El subsistema de ingesta batch se encarga de leer los ficheros CSV del directorio landing y cargarlos en la capa Bronze. Utiliza Spark SQL para la lectura y escritura en formato Delta Lake, añadiendo metadatos de ingesta como la fecha de carga y el fichero de origen.

El subsistema de ingesta streaming gestiona el flujo de pedidos en tiempo real. Un simulador desarrollado en Python genera eventos de pedido con datos sintéticos realistas y los publica en un topic de Kafka. Un job de Spark Structured Streaming consume estos eventos y los persiste en Delta Lake con checkpointing para garantizar la semántica exactly-once.

El subsistema de procesamiento implementa las transformaciones de la capa Silver y la construcción del modelo dimensional en Gold. Todos los procesos se ejecutan con PySpark, lo que permite aprovechar el procesamiento distribuido y mantener la coherencia en el tratamiento de datos batch y streaming.

El subsistema de machine learning integra dos modelos desarrollados con Spark MLlib. El modelo de clustering utiliza K-Means para segmentar los clientes según su comportamiento de compra. El modelo de forecasting emplea Gradient Boosted Trees para predecir la demanda mensual por producto.

El subsistema de transformaciones analíticas utiliza dbt para generar los marts finales. Estos marts agregan y transforman los datos del modelo dimensional para facilitar el análisis de rendimiento comercial, geográfico y de producto.

El subsistema de orquestación coordina la ejecución de todos los componentes mediante Apache Airflow. El DAG principal define las dependencias entre tareas y garantiza que se ejecuten en el orden correcto. Airflow proporciona también capacidades de monitorización, logging y reintentos automáticos.

Finalmente, el subsistema de almacenamiento utiliza Delta Lake como formato unificado para todas las capas. Esto proporciona transacciones ACID, versionado de datos (time travel), optimización automática y compatibilidad con el ecosistema Spark.

5.3. Justificación de elección de tecnologías (bases de datos, herramientas de procesamiento, orquestación, etc.)

La selección de tecnologías responde a criterios de idoneidad técnica, madurez del ecosistema, alineación con las prácticas de la industria y viabilidad para un despliegue local.

Apache Spark se eligió como motor de procesamiento por su capacidad para manejar tanto cargas batch como streaming con un API unificado, su escalabilidad horizontal y su amplia adopción en el sector. Spark permite escribir transformaciones complejas en Python (PySpark) manteniendo un rendimiento óptimo gracias a su motor de ejecución distribuido.

Delta Lake se seleccionó como formato de almacenamiento por sus capacidades ACID, esenciales para garantizar la consistencia de los datos en operaciones de escritura concurrentes. Además, Delta Lake es el formato nativo de Databricks, lo que facilita una eventual migración a esa plataforma. Su compatibilidad con Parquet permite integración con otras herramientas del ecosistema.

Apache Kafka se incorporó para la gestión de eventos en tiempo real por ser el estándar de la industria para streaming de datos. Su modelo de topics y particiones ofrece alta disponibilidad, durabilidad de mensajes y capacidad para desacoplar productores y consumidores.

Apache Airflow se eligió para orquestación por su flexibilidad en la definición de workflows como código Python (DAGs), su interfaz web para monitorización y su amplio ecosistema de operadores y conexiones. Es una herramienta madura y muy utilizada en entornos de producción de data engineering.

dbt (data build tool) se incorporó para la capa de transformaciones analíticas por su enfoque de transformaciones como código SQL, su sistema de testing integrado y su capacidad para generar documentación automática. dbt permite aplicar prácticas de software engineering al desarrollo de modelos de datos.

Docker y Docker Compose se utilizaron para contenerizar todos los servicios, garantizando la reproducibilidad del entorno y simplificando el despliegue. Esta aproximación permite que cualquier persona pueda levantar el sistema completo con un único comando, sin necesidad de instalar dependencias individualmente.

Componente	Tecnología	Versión
Procesamiento	Apache Spark	3.5.1
Almacenamiento	Delta Lake	3.1.0
Streaming	Apache Kafka	4.1.1
Orquestación	Apache Airflow	2.10.4
Transformaciones	dbt Core	1.11.2
ML	Spark MLlib	3.5.1
Contenedores	Docker Compose	v2
Lenguaje	Python	3.11
Base de datos	PostgreSQL	15
Visualización y dashboard	Power BI Desktop	2.150.2102.0

Conviene subrayar que la selección tecnológica final mantiene plena coherencia con la propuesta original del proyecto. Donde antes se planteaba Azure Databricks, se utiliza Apache Spark con la misma API de DataFrames y el mismo soporte para Delta Lake. Donde se proponía Databricks Jobs para la orquestación, se emplea Apache Airflow, una herramienta más versátil y con mayor adopción fuera del ecosistema Databricks. Las transformaciones que se habrían implementado como notebooks en Databricks se materializan ahora mediante dbt, lo que añade testing declarativo y documentación automática que el planteamiento original no contemplaba. En

conjunto, el stack final no solo iguala las capacidades de la propuesta cloud, sino que en algunos aspectos la supera al incorporar herramientas estándar de la industria que enriquecen el perfil técnico del proyecto.

5.4. Estimación básica de costes de infraestructura

Como se ha explicado en la introducción, el proyecto se desarrollo inicialmente sobre Azure Databricks con créditos académicos. El agotamiento de dichos créditos motivo la migración a un entorno local con Docker, lo que implica un coste de infraestructura cloud nulo durante la fase final del desarrollo. No obstante, se presenta a continuación una estimación orientativa de lo que supondría ejecutar una carga de trabajo similar en un entorno cloud, tanto para contextualizar la decisión tomada como para facilitar una eventual migración.Para un volumen de datos como el del proyecto (aproximadamente 250.000 registros con actualización diaria), una configuración mínima en Azure Databricks incluiría un cluster de desarrollo con 2 nodos Standard_DS3_v2, lo que supondría un coste aproximado de 150-200 euros mensuales en horario laboral.

Si se optara por servicios gestionados equivalentes en AWS, la combinación de EMR para Spark, MSK para Kafka y MWAA para Airflow podría situarse en el rango de 300-400 euros mensuales para un entorno de desarrollo.

El enfoque local adoptado demuestra que es posible desarrollar y validar arquitecturas de datos modernas sin incurrir en costes significativos, reservando la inversión en cloud para entornos de producción con volúmenes de datos reales.

5.5. Estrategia DevOps

Aunque el foco principal del proyecto es la ingeniería de datos y no las prácticas DevOps, se han incorporado elementos que facilitan la operación y mantenibilidad del sistema.

El control de versiones se gestiona con Git, con el repositorio alojado en GitHub. El código se organiza en directorios según su función (ingesta, transformaciones, ML, orquestación), facilitando la navegación y el mantenimiento.

La contenerización con Docker permite recrear el entorno de desarrollo de forma determinista. El fichero docker-compose.yml define todos los servicios necesarios y sus dependencias, de modo que el comando docker compose up levanta la infraestructura completa.

Los tests de calidad de datos se implementan en dbt, ejecutándose automáticamente como parte del pipeline de Airflow. Esto garantiza que cualquier problema de calidad se detecte antes de que los datos lleguen a las capas de consumo.

La documentación técnica incluye un README con instrucciones de despliegue, un runbook de operaciones con comandos útiles y troubleshooting, y un diccionario de datos con la descripción de todas las tablas y columnas.

6. Solución tecnológica

6.1. Fuentes de datos

El proyecto utiliza dos fuentes de datos principales que alimentan el sistema de forma complementaria.

La fuente principal es un dataset público de ventas farmacéuticas obtenido de Kaggle (pharma-data.csv). Este dataset contiene aproximadamente 250.000 registros de transacciones comerciales con información detallada sobre productos vendidos, clientes (farmacias y hospitales), representantes comerciales, ubicaciones geográficas, canales de venta y valores monetarios. El dataset cubre un periodo de seis años y ocho países europeos, lo que permite análisis temporales y geográficos significativos.

La segunda fuente es un generador de eventos sintéticos desarrollado en Python que simula pedidos en tiempo real. Este simulador utiliza la librería Faker para generar datos realistas y publica los eventos en un topic de Kafka. Los pedidos sintéticos incluyen identificadores únicos, timestamps, productos, cantidades, precios y ubicaciones, siguiendo una distribución que simula patrones reales de comportamiento comercial.

6.2. Preparación de datos

La preparación de datos se realiza en la capa Silver, donde se aplican transformaciones de limpieza, normalización y validación.

Las transformaciones de limpieza incluyen la eliminación de espacios en blanco superfluos, la conversión de valores nulos a valores por defecto cuando procede, y la estandarización de formatos de texto (mayúsculas, minúsculas, capitalización).

La normalización abarca la conversión de tipos de datos (strings a numéricos, fechas a formato estándar), la unificación de valores categóricos (por ejemplo, códigos de país a nombres completos) y la generación de columnas derivadas como el importe total de cada transacción.

Las validaciones implementadas verifican que las cantidades sean positivas, que los precios no sean negativos, que los campos obligatorios contengan valores y que las coordenadas geográficas estén dentro de rangos válidos. Los registros que no superan las validaciones críticas se derivan a una tabla de rechazados para su posterior análisis.

La deduplicación se aplica tanto a los datos batch (por clave natural compuesta) como a los datos de streaming (por identificador de evento), garantizando que no haya registros duplicados en las capas de consumo.

6.3. Flujos de datos

El sistema implementa dos flujos de datos diferenciados que convergen en la capa Gold.

El flujo batch procesa los datos históricos de ventas siguiendo la secuencia: landing (CSV) -> Bronze (raw) -> Silver (curated) -> Gold (dimensional). Cada etapa genera tablas Delta Lake independientes, lo que permite reprocesar cualquier capa sin afectar a las anteriores. Este flujo se ejecuta de forma programada mediante Airflow.

El flujo streaming procesa los pedidos en tiempo real siguiendo la secuencia: Kafka (topic orders) -> Bronze (orders_stream_raw) -> Silver (orders) -> Gold (fact_orders). El consumidor de Spark Structured Streaming utiliza checkpointing para garantizar la semántica exactly-once y evitar pérdida o duplicación de eventos ante fallos.

Ambos flujos alimentan el modelo dimensional de Gold, donde se integran para ofrecer una visión unificada del negocio que combina datos históricos con información en tiempo real.

6.4. Procesamiento (batch, streaming, etc.)

El procesamiento batch se implementa mediante scripts PySpark que leen datos de una capa, aplican transformaciones y escriben el resultado en la capa siguiente. Las transformaciones se definen de forma declarativa utilizando el API de DataFrames de Spark, lo que permite optimización automática por parte del motor de ejecución.

Las operaciones típicas incluyen selección y renombrado de columnas, filtrado de registros, agregaciones, joins entre datasets y funciones de ventana para cálculos como rankings y acumulados. El uso de Delta Lake permite operaciones de merge (upsert) para actualizaciones incrementales.

El procesamiento streaming utiliza Spark Structured Streaming con el conector de Kafka. El consumidor define un esquema para los eventos JSON, aplica transformaciones equivalentes a las del flujo batch y escribe en Delta Lake en modo append con checkpointing. El trigger se configura en modo processingTime para procesar micro-batches cada pocos segundos.

La unificación de ambos paradigmas bajo Spark permite reutilizar lógica de transformación y mantener consistencia en el tratamiento de los datos independientemente de su origen.

6.5. Orquestación

La orquestación del pipeline se gestiona mediante Apache Airflow, que coordina la ejecución de todas las tareas en el orden correcto y con las dependencias adecuadas.

El DAG principal (lakehouse_farma_pipeline) organiza las tareas en grupos lógicos que corresponden a las capas de la arquitectura:

El grupo bronze_layer contiene la tarea de ingesta del CSV de ventas. Esta tarea lee el fichero del directorio landing y lo escribe en formato Delta Lake en la capa Bronze.

El grupo silver_layer contiene las tareas de curación de datos: curate_pharma_sales y curate_orders. Estas tareas se ejecutan en paralelo ya que no tienen dependencias entre sí.

El grupo gold_layer contiene la tarea build_dimensional_model, que construye las dimensiones y las tablas de hechos a partir de los datos curados.

El grupo ml_layer contiene las tareas de machine learning: customer_clustering y demand_forecast. Estas tareas también se ejecutan en paralelo, ya que ambas parten de los mismos datos de Gold pero generan resultados independientes.

El grupo dbt_layer contiene las tareas de transformación analítica: dbt_run para ejecutar los modelos y dbt_test para validar la calidad de los datos resultantes.

Cada tarea se implementa como un SparkSubmitOperator o BashOperator según corresponda. Se configuran reintentos automáticos con backoff exponencial para manejar fallos transitorios. El DAG se puede ejecutar manualmente desde la interfaz web de Airflow o programarse para ejecución periódica.

6.6. Almacenamiento y consulta

El almacenamiento se basa íntegramente en Delta Lake, un formato de tabla abierto que extiende Parquet con capacidades de gestión transaccional.

Cada tabla Delta se almacena como un directorio que contiene ficheros Parquet con los datos y un subdirectorio _delta_log con el registro de transacciones. Este registro permite funcionalidades avanzadas como time travel (consultar versiones anteriores), rollback, y optimización automática de ficheros.

La organización física de los datos sigue la estructura de capas del patrón medallion: /data/bronze, /data/silver y /data/gold. Dentro de cada capa, cada tabla tiene su propio directorio. Esta organización facilita la gestión de permisos, backups y limpieza de datos obsoletos.

Para consultar los datos se utiliza Spark SQL a través del Thrift Server, que expone una interfaz JDBC/ODBC compatible con herramientas como DBeaver, Tableau o cualquier cliente SQL estándar. Las tablas se registran en el metastore de Spark, lo que permite referenciarlas por nombre en las consultas.

El rendimiento de las consultas se beneficia de las optimizaciones de Delta Lake, incluyendo data skipping (omitir ficheros que no contienen datos relevantes basándose en estadísticas), Z-ordering (colocación de datos relacionados) y caching automático de metadatos.

6.7. Explotación de resultados (reporting, modelos…)

La explotación de resultados se articula en torno a tres elementos principales: el modelo dimensional, los modelos de machine learning y los marts analíticos.

El modelo dimensional sigue un esquema en estrella clásico con cuatro dimensiones y dos tablas de hechos. La dimensión dim_customer contiene los datos maestros de los clientes (nombre, ubicación, canal). La dimensión dim_product almacena el catálogo de productos con su clasificación terapéutica. La dimensión dim_sales_rep recoge la jerarquía comercial (representante, manager, equipo). La dimensión dim_time proporciona los atributos temporales (mes, trimestre, año). Las tablas de hechos fact_sales y fact_orders contienen las métricas transaccionales (cantidad, precio, importe) con claves foráneas a las dimensiones.

Los modelos de machine learning generan dos tablas adicionales en Gold. La tabla ml_customer_clusters contiene la asignación de cada cliente a uno de los tres segmentos identificados (Premium, Regular, Básico), junto con las métricas que caracterizan su comportamiento. La tabla ml_demand_forecast contiene las predicciones de demanda mensual por producto, incluyendo los valores reales para comparación y las métricas de error.

Los marts analíticos generados con dbt proporcionan vistas agregadas listas para consumo:

Mart	Descripción	Métricas principales
mart_sales_rep_performance	Rendimiento comercial	Ventas totales, clientes, transacciones
mart_geographic_analysis	Análisis geográfico	Ventas por país/ciudad, ticket medio
mart_product_performance	Rendimiento productos	Unidades, ingresos, clientes
mart_cluster_analysis	Segmentos clientes	Tamaño cluster, ventas medias
mart_forecast_analysis	Precisión forecast	MAPE, accuracy por producto

Para completar la capa de consumo y facilitar la explotación de los datos por parte de usuarios de negocio, se ha desarrollado un dashboard interactivo en Power BI Desktop. Este dashboard se conecta directamente al Spark Thrift Server mediante el conector JDBC nativo (jdbc:hive2://localhost:10000), lo que permite consultar en tiempo real las tablas Gold y los marts analíticos registrados en el metastore de Spark sin necesidad de extracciones manuales.

El dashboard se estructura en tres páginas temáticas. La primera, Resumen Ejecutivo, presenta los KPIs principales del negocio (ventas totales, número de transacciones, clientes únicos, ticket medio y número de productos), acompañados de gráficos de ventas por clase de producto, por distribuidor y por canal de venta, junto con un segmentador temporal que permite filtrar por año. La segunda página, Marts dbt, expone directamente los resultados de los marts analíticos generados por dbt Core: la tabla de segmentos de clientes con sus métricas agregadas, la precisión del modelo de forecast por producto con su categoría de accuracy, y los rankings de los diez mejores comerciales y productos por volumen de ventas. La tercera página, Machine Learning, visualiza los resultados de ambos modelos: las métricas clave (Silhouette Score, clientes premium, accuracy promedio y R^2 del modelo), la distribución de clientes en los tres segmentos identificados por K-Means, las ventas por segmento, la precisión del forecast por producto y un gráfico de dispersión que compara las ventas reales con las predichas.

El dashboard utiliza un tema personalizado con la paleta corporativa del proyecto y medidas DAX que permiten cálculos dinámicos como el ticket medio, adaptándose automáticamente a los filtros aplicados. Las capturas de pantalla de las tres páginas se incluyen en el Anexo E.

6.8. Consideraciones éticas / legales del uso de datos

El proyecto no plantea implicaciones éticas ni legales significativas en materia de protección de datos. La fuente principal es un dataset público disponible en Kaggle bajo licencia abierta, que contiene datos comerciales agregados sin información personal identificable. Los datos de streaming son completamente sintéticos, generados por un simulador desarrollado específicamente para este proyecto mediante la librería Faker, por lo que no existe ninguna vinculación con personas o entidades reales. En consecuencia, el tratamiento de datos realizado no está sujeto a las obligaciones del Reglamento General de Protección de Datos (RGPD) ni requiere medidas adicionales de anonimización o consentimiento.

7. Resultados y conclusiones

7.1. Logros alcanzados y validación de los resultados

El proyecto ha cumplido satisfactoriamente todos los objetivos planteados inicialmente, resultando en un sistema de datos funcional y completo.

Se ha construido una arquitectura lakehouse operativa que implementa el patrón medallion con tres capas diferenciadas. Los datos fluyen correctamente desde las fuentes hasta las tablas de consumo, con transformaciones documentadas y trazables en cada etapa.

Se han implementado con éxito ambos modos de procesamiento: batch para los datos históricos y streaming para los pedidos en tiempo real. El sistema demuestra la viabilidad de combinar ambos paradigmas bajo una arquitectura unificada.

El modelo dimensional en estrella facilita consultas analíticas eficientes. Las cuatro dimensiones y dos tablas de hechos proporcionan la flexibilidad necesaria para responder a preguntas de negocio variadas.

Los dos modelos de machine learning aportan valor analítico demostrable. El clustering identifica tres segmentos de clientes con características diferenciadas y accionables. El modelo de forecasting predice la demanda con precisión suficiente para la mayoría de productos.

La capa de testing con dbt valida automáticamente la calidad de los datos. Los 21 tests implementados cubren aspectos como nulos, unicidad y valores permitidos, proporcionando confianza en la integridad de la información.

Complementariamente, el proyecto incluye 55 tests unitarios implementados con pytest que validan las funciones puras del pipeline, la consistencia de configuraciones entre scripts, el esquema de eventos de Kafka y la correcta carga del catálogo de datos del simulador. Estos tests utilizan técnicas como monkeypatch para simular variables de entorno, fixtures con ficheros CSV temporales y mocker.spy para verificar la invocación de métodos internos, y se ejecutan en menos de un segundo sin necesidad de levantar el clúster de Spark.

La orquestación con Airflow coordina todo el pipeline de forma fiable. El DAG se ejecuta correctamente de principio a fin en aproximadamente 7 minutos, con capacidad de reintentos y monitorización.

La contenerización con Docker garantiza la reproducibilidad. El sistema puede desplegarse en cualquier máquina con Docker instalado mediante un único comando, sin configuraciones adicionales.

Adicionalmente, Power BI Desktop actúa como capa de visualización, conectándose al Spark Thrift Server vía JDBC para consumir tanto las tablas Gold como los marts analíticos generados por dbt.

7.2. Métricas utilizadas

Las métricas del proyecto se organizan en tres categorías: volumetría de datos, calidad del modelo de ML y rendimiento del sistema.

En cuanto a volumetría, el sistema procesa 249.704 registros de ventas históricas, gestiona 751 clientes únicos, 241 productos distintos y cubre 8 países. El modelo dimensional genera aproximadamente 250.000 registros en la tabla de hechos principal.

Respecto a los modelos de machine learning, el clustering de clientes alcanza un Silhouette Score de 0.817, lo que indica una separación clara entre los tres segmentos identificados. El modelo de forecasting obtiene precisiones que varían entre el 87% y el 99% según el producto, con una media ponderada superior al 90%.

En términos de rendimiento, el pipeline completo se ejecuta en aproximadamente 7 minutos. La capa de dbt ejecuta 6 modelos y 21 tests de calidad de datos en menos de 30 segundos. Adicionalmente, los 55 tests unitarios con pytest se ejecutan en menos de un segundo. El consumo de memoria del stack completo de Docker se mantiene por debajo de 8GB, compatible con equipos de desarrollo estándar.

Métrica	Valor
Registros procesados	249.704
Clientes segmentados	751
Productos	241
Países	8
Silhouette Score (clustering)	0.817
Precisión forecast (media)	>90%
Tiempo ejecución pipeline	~7 minutos
Tests dbt pasando	21/21
Modelos dbt	6
Tests unitarios pytest	55/55

7.3. Limitaciones identificadas

A pesar de los resultados positivos, el proyecto presenta algunas limitaciones que es importante señalar.

La limitación más evidente es el cambio de plataforma respecto a la propuesta original. Aunque la implementación local con Docker demuestra las mismas competencias técnicas y produce resultados equivalentes, no permite evaluar ciertos aspectos que solo se manifiestan en un entorno cloud, como la gestión de permisos con Unity Catalog, la monitorización con servicios gestionados o el comportamiento del sistema bajo elasticidad automática de recursos. Sin embargo, el código y la arquitectura están diseñados para que esta migración sea directa: los scripts de Spark funcionan sin modificaciones en Databricks, y las tablas Delta Lake son compatibles nativamente con el formato de Databricks.

La siguiente limitación notable es el carácter sintético de los datos de streaming. Aunque el simulador genera eventos realistas, no captura toda la complejidad de un flujo de pedidos real, como patrones estacionales, picos de demanda impredecibles o errores en los datos de origen.

El modelo dimensional, aunque funcional, no implementa todas las técnicas avanzadas de modelado como Slowly Changing Dimensions (SCD) tipo 2 para historificar cambios en las dimensiones. Actualmente las dimensiones reflejan solo el estado actual.

El modelo de forecasting, al utilizar solo features temporales, no incorpora variables exógenas que podrían mejorar las predicciones, como campañas de marketing, estacionalidad de enfermedades o eventos externos.

La infraestructura local, aunque suficiente para el desarrollo y demostración, no permite evaluar el comportamiento del sistema bajo volúmenes de datos significativamente mayores o con requisitos de latencia estrictos.

En cuanto a la capa de visualización, se ha optado por Power BI Desktop como herramienta de dashboarding, conectado via JDBC al Spark Thrift Server. Aunque esta solución cumple con el objetivo de demostrar la explotación de los datos, en un entorno productivo sería recomendable utilizar una herramienta open source como Apache Superset o Metabase que permita el acceso concurrente de múltiples usuarios sin necesidad de licencias individuales.

7.4. Futuras líneas de mejora y/o desarrollo

El proyecto sienta las bases para múltiples líneas de evolución que podrían abordarse en trabajos futuros.

La línea de trabajo más inmediata sería la migración del sistema a un entorno cloud, retomando el planteamiento original sobre Azure Databricks. Esta migración requeriría cambios mínimos en el código de procesamiento, ya que Apache Spark y Delta Lake son nativos en esa plataforma. La orquestación con Airflow podría sustituirse por Databricks Jobs o mantenerse mediante Azure Managed Airflow. Los modelos de dbt funcionarían sobre la misma conexión Spark o podrían migrarse al adaptador nativo de Databricks. Esta portabilidad es una de las ventajas deliberadas del diseño adoptado.

La migración de la capa de visualización desde Power BI Desktop hacia una herramienta open source y auto-alojada como Apache Superset o Metabase permitiría el acceso concurrente de múltiples usuarios de negocio sin dependencia de licencias Microsoft, democratizando el acceso a los insights generados por el sistema en un entorno productivo.

La implementación de SCD tipo 2 en las dimensiones permitiría mantener el histórico de cambios y realizar análisis as-was, comparando métricas con el estado de las dimensiones en diferentes momentos del tiempo.

La mejora del modelo de forecasting podría incluir variables exógenas, modelos más sofisticados como Prophet o redes neuronales recurrentes, y un sistema de retroalimentación que reentrene el modelo periódicamente con datos recientes.

La implementación de un catálogo de datos con herramientas como DataHub o Amundsen mejoraría la descubribilidad y gobernanza de los activos de datos.

Finalmente, la integración con un sistema de CI/CD automatizaría el despliegue de cambios y garantizaría que el código pase los tests antes de llegar a producción.

8. Bibliografía

Armbrust, M., et al. (2020). Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proceedings of the VLDB Endowment*.

Chambers, B., y Zaharia, M. (2018). *Spark: The Definitive Guide*. O'Reilly Media.

Databricks. (2021). Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. *CIDR Conference*.

Kimball, R., y Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley.

Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.

Kreps, J., Narkhede, N., y Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. *NetDB Workshop*.

Lekshmi Narayanan, S. (2020). Pharma Sales Data [Dataset]. Kaggle. https://www.kaggle.com/datasets/lekshminnarayanan/pharma-sales-data

Apache Airflow Documentation. https://airflow.apache.org/docs/

Apache Kafka Documentation. https://kafka.apache.org/documentation/

Apache Spark Documentation. https://spark.apache.org/docs/latest/

dbt Documentation. https://docs.getdbt.com/

Delta Lake Documentation. https://docs.delta.io/

Docker Documentation. https://docs.docker.com/

Kaggle Dataset: https://www.kaggle.com/datasets/milanzdravkovic/pharma-sales-data

9. Anexos

Anexo A: Estructura del repositorio

El repositorio del proyecto se organiza en los siguientes directorios principales:

```
tfm-lakehouse-farma/
├── airflow/
│   └── dags/                    # DAG de orquestación (Airflow)
├── data/
│   ├── export/                  # CSVs exportados de Gold y Marts
│   └── landing/                 # Datos fuente (CSV)
├── dbt_gold/
│   └── models/
│       ├── staging/             # Modelo staging (vista)
│       └── marts/               # Marts analíticos (tablas)
├── src/
│   ├── simulator/               # Generador de eventos Kafka
│   └── spark/                   # Scripts de procesamiento Spark
├── tests/                       # Tests unitarios (pytest)
├── docs/                        # Documentación técnica
├── docker-compose.yml           # Definición de servicios
├── requirements-dev.txt         # Dependencias de test
├── pytest.ini                   # Configuración pytest
├── .env                         # Variables de entorno
├── .gitignore                   # Exclusiones de git
└── README.md                    # Instrucciones de uso
```

Anexo B: Comandos principales

Levantar el entorno:

```
docker compose up -d
```

Ejecutar el pipeline desde Airflow:

1. Acceder a http://localhost:8080
2. Iniciar sesión con credenciales airflow/airflow
3. Activar el DAG lakehouse_farma_pipeline
4. Hacer clic en Trigger DAG

Ejecutar dbt manualmente:

```
cd dbt_gold

dbt run --profiles-dir .

dbt test --profiles-dir .
```

Ejecutar tests unitarios:

```
pip install -r requirements-dev.txt

pytest tests/ -v
```

Consultar datos con Beeline:

```
docker exec -it tfm-lakehouse-farma-spark-thrift-1 \

  /opt/spark/bin/beeline -u "jdbc:hive2://localhost:10000"
```

Apagar el entorno:

```
docker compose down
```

Anexo C: Diccionario de datos resumido

Tabla	Layer	Descripción	Registros
pharma_sales_raw	Bronze	Ventas crudas del CSV	~250K
orders_stream_raw	Bronze	Pedidos streaming crudos	Variable
pharma_sales	Silver	Ventas limpias y validadas	~250K
orders	Silver	Pedidos deduplicados	Variable
dim_customer	Gold	Dimensión clientes	751
dim_product	Gold	Dimensión comercial	241
dim_sales_rep	Gold	Dimensión comercial	~80
dim_time	Gold	Dimensión tiempo	72
fact_sales	Gold	Hechos ventas	~250k
fact orders	Gold	Hechos pedidos	Variable
ml_customer_clusters	Gold	Segmentos clientes	751
ml_demand_forecast	Gold	Predicciones demanda	~2800

Anexo D: Enlace al repositorio y vídeo

Repositorio GitHub: https://github.com/ebarba21/tfm-lakehouse-farma

Video demostrativo: https://youtu.be/yLMZCr420Go

Nota: El acceso al repositorio se proporcionará a los tutores para la evaluación.

Anexo E: Dashboard Power BI

Figura E.1: Resumen Ejecutivo — KPIs principales (ventas totales, número de transacciones, clientes únicos, ticket medio y número de productos), ventas por clase de producto, ventas por distribuidor y distribución por canal de venta. Incluye segmentador temporal por año.

Figura E.2: KPIs dbt CORE - MART — Resultados de los marts analíticos generados por dbt Core: tabla de segmentos de clientes (mart_cluster_analysis) con métricas agregadas por segmento, tabla de precisión del modelo de forecast por producto (mart_forecast_analysis) con categoría de accuracy, y rankings de los diez mejores comerciales (mart_sales_rep_performance) y productos (mart_product_performance) por volumen de ventas.

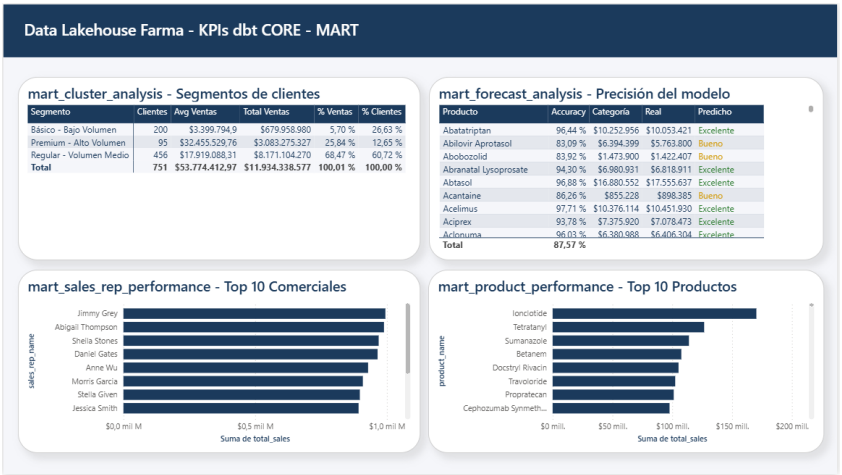


Figura E.3: Machine Learning — Segmentación (K-Means) y Forecast Demanda (GBT Regressor) — KPIs de los modelos (Silhouette Score 0,817; clientes premium; accuracy promedio 87,57%; R^2 0,86), distribución de clientes en los tres segmentos identificados, ventas totales por segmento, precisión del forecast por producto y gráfico de dispersión comparando ventas reales con predichas.