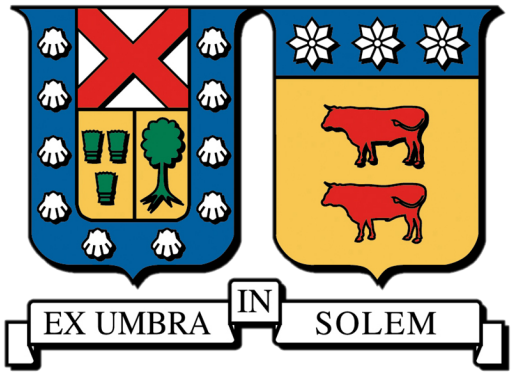
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
SANTIAGO - CHILE



"PLATAFORMA PARA LA GESTIÓN DE MARKET DATA EN TIEMPO REAL"

GIANFRANCO ROLLA TENEOS

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL TELEMÁTICO

PROFESOR GUÍA: **Patricio Olivares**

PROFESOR CORREFERENTE: **Nicolás Torres**

ENERO - 2025

Agradecimientos

Quiero expresar mi sincero agradecimiento a mi profesor guía y a los docentes que me acompañaron a lo largo de mi formación académica. Su dedicación y disposición para compartir sus conocimientos fueron fundamentales no solo para superar los desafíos de este proyecto, sino también para mi crecimiento profesional y personal durante toda mi carrera.

A mi familia, gracias por su apoyo incondicional y por brindarme un hogar lleno de tranquilidad y confianza. Su paciencia, comprensión y el espacio que me dieron para concentrarme fueron fundamentales para que pudiera llegar hasta aquí.

A mis amigos y compañeros de estudio, gracias por el apoyo mutuo, las conversaciones que nos ayudaron a resolver problemas y por estar ahí en los momentos difíciles y en las victorias. Haber compartido esta experiencia juntos fue clave para llegar hasta aquí.

Finalmente, agradezco a todas las personas que, de una u otra forma, contribuyeron al desarrollo de esta tesis. Su ayuda y colaboración fueron fundamentales para alcanzar este logro.

Este documento está dedicado a mi perro, Blacky, quien, aunque no entiende nada de este trabajo, estuvo a mi lado como el mejor compañero en las largas noches de estudio. Sus paseos diarios fueron clave para despejar la mente y recargar energías para seguir adelante.

Resumen

Este proyecto presenta el desarrollo de una plataforma para la gestión y análisis de grandes volúmenes de datos en tiempo real, diseñada para abordar los desafíos de almacenamiento y procesamiento en áreas de alta demanda como el ámbito financiero, científico y tecnológico. La solución propuesta permite manejar datos de alta frecuencia de manera escalable, optimizando la toma de decisiones en contextos donde el tiempo de respuesta es crítico.

El trabajo se centra en la implementación de un módulo de ingesta y procesamiento de información en tiempo real, enfocado en el sector financiero y, específicamente, en el mercado de criptomonedas, utilizando datos de la plataforma Binance. Para el almacenamiento se emplea MongoDB, y el procesamiento se realiza con tecnologías como Apache Kafka y Apache Spark, que permiten administrar y analizar los datos mediante Procesamiento en Streaming y batch.

Los resultados obtenidos muestran que el sistema es capaz de realizar la ingesta y procesamiento de datos en tiempo real de manera eficiente, permitiendo el análisis y almacenamiento de la información en MongoDB. El uso de tecnologías como Apache Kafka y Apache Spark facilita el manejo de grandes volúmenes de datos y asegura la disponibilidad de estos para futuras consultas y análisis.

Palabras clave: Big Data, Procesamiento en Streaming, Apache Kafka, Apache Spark, Análisis en Tiempo Real, Criptomonedas, Binance, MongoDB.

Abstract

This project presents the development of a platform for the management and analysis of large volumes of real-time data, designed to address storage and processing challenges in high-demand areas such as finance, science and technology. The proposed solution enables the handling of high-frequency data in a scalable manner, optimizing decision-making processes in contexts where response time is critical.

The work focuses on the implementation of a real-time data ingestion and processing module, specifically targeting the financial sector and cryptocurrency markets using data from the Binance platform. MongoDB is used for storage, and the processing is performed using technologies such as Apache Kafka and Apache Spark, enabling data management and analysis through streaming and batch processing.

The results demonstrate that the system efficiently ingests and processes real-time data, enabling the analysis and storage of information in MongoDB. Technologies such as Apache Kafka and Apache Spark facilitate the handling of large volumes of data and ensure data availability for future queries and analyses.

Palabras clave: Big Data, Streaming Data Processing, Apache Kafka, Apache Spark, Real-time Data Analysis, Cryptocurrencies, Binance, MongoDB.

Glosario

Análisis en Tiempo Real Proceso de interpretación y procesamiento de datos a medida que estos son generados o recibidos, con el objetivo de producir información útil o acciones en el menor tiempo posible. Es clave en aplicaciones donde la inmediatez es crítica, como en el análisis de mercados financieros o la detección de eventos en sistemas de monitoreo.

Apache Kafka Plataforma de mensajería distribuida que permite la ingesta de grandes volúmenes de datos en tiempo real. Es utilizada para transmitir y procesar flujos de datos en sistemas de alto rendimiento.

Apache Spark Framework de procesamiento de datos en clústeres que permite realizar análisis de grandes volúmenes de datos. En este proyecto, se utiliza para el procesamiento en streaming y batch, facilitando el análisis de datos en tiempo real.

Big Data Grandes volúmenes de datos que son complejos, variados y que se generan a alta velocidad, superando la capacidad de las herramientas tradicionales para capturarlos, gestionarlos y procesarlos de manera eficiente.

Binance API Interfaz de programación de aplicaciones (API) proporcionada por Binance, que permite acceder a datos de mercado de criptomonedas en tiempo real y realizar transacciones en la plataforma.

Data Lake Repositorio centralizado de almacenamiento de datos en su forma bruta, estructurada y no estructurada, que permite la escalabilidad y la flexibilidad en el análisis de datos.

Formato de Datos Estructura en la cual los datos están organizados y almacenados para ser interpretados adecuadamente por el sistema. En el contexto de este proyecto, el formato de datos es JSON, que permite una representación clara y organizada de la información de las transacciones y precios recibidos.

Fuente de Datos Origen o proveedor de donde se obtienen los datos utilizados en el sistema. En este proyecto, la fuente de datos principal es el exchange Binance, desde donde se recibe market data en tiempo real para su procesamiento y análisis.

Ingesta de Datos Proceso de capturar y almacenar datos desde múltiples fuentes en un sistema centralizado para su posterior análisis y procesamiento.

JSON (JavaScript Object Notation) Es un formato de intercambio de datos ligero y legible, utilizado comúnmente para almacenar y transmitir datos en aplicaciones web.

Market Data Información financiera que incluye precios, volúmenes, y transacciones relacionadas con activos en mercados financieros. En este proyecto, representa los datos recopilados en tiempo real desde el exchange Binance.

MongoDB Base de datos NoSQL orientada a documentos que permite almacenar datos de manera flexible y escalable. MongoDB es utilizada en este proyecto como almacenamiento principal para los datos procesados.

NoSQL Tipo de base de datos diseñada para manejar grandes volúmenes de datos no estructurados o semi-estructurados. Es conocida por su flexibilidad y escalabilidad, y utiliza esquemas dinámicos en lugar de tablas relacionales tradicionales.

Offset Es un identificador único asociado a cada mensaje o registro en un sistema de procesamiento o almacenamiento de datos. Permite realizar un seguimiento pre-

ciso del orden y la ubicación de los datos dentro de un flujo o conjunto, facilitando la recuperación secuencial y el manejo eficiente de los eventos.

Procesamiento en Batch Técnica de procesamiento en la que los datos son acumulados y procesados en grupos o lotes en momentos específicos.

Procesamiento en Streaming Técnica de procesamiento en tiempo real que permite analizar y procesar los datos a medida que se generan.

RDD (Resilient Distributed Dataset) Es una estructura de datos fundamental en Apache Spark que permite almacenar y procesar colecciones de datos de manera distribuida y tolerante a fallos. Es ideal para operaciones como mapeos, filtros y reducciones sobre grandes volúmenes de datos.

Spring Framework Framework de desarrollo de aplicaciones en Java que facilita la creación de aplicaciones robustas y escalables. En este proyecto, Spring se utiliza para gestionar las conexiones entre los componentes y la integración con MongoDB.

Tick by Tick Método de registro que captura cada cambio individual en el precio, volumen o estado del mercado en tiempo real, proporcionando un nivel de detalle granular para el análisis financiero.

Acrónimos

API Application Programming Interface.

CLI Command Line Interface.

GUI Graphical User Interface.

I/O Input/Output.

JVM Java Virtual Machine.

KPI Key Performance Indicator.

RDD Resilient Distributed Dataset.

SDK Software Development Kit.

SQL Structured Query Language.

Tabla de Contenido

Índice de figuras

Índice de tablas

1. Introducción

En la última década, el avance de la tecnología y la creciente capacidad de los sistemas para manejar grandes volúmenes de datos han transformado radicalmente múltiples industrias, y el sector financiero no es la excepción. Con el surgimiento del *Big Data*, los mercados financieros han experimentado un cambio hacia la recolección y análisis de datos en tiempo real para optimizar la toma de decisiones y mejorar la competitividad. En este contexto, el *market data tick by tick*, que captura cada variación en el precio, el volumen de transacciones y las actualizaciones en el libro de órdenes, se ha convertido en una herramienta fundamental para las corredoras de bolsa y otros actores del mercado.

La cantidad y velocidad a la que se generan estos datos suponen un desafío significativo para su gestión. Las tecnologías tradicionales de almacenamiento y procesamiento no están diseñadas para soportar el volumen ni la frecuencia con la que se necesita capturar, almacenar y analizar esta información. Es aquí donde los *Data Lakes* y tecnologías de procesamiento en tiempo real como Apache Kafka y Apache Spark juegan un rol crucial. Estas herramientas permiten manejar de manera eficiente los datos de alta frecuencia, ofreciendo a las instituciones financieras una infraestructura capaz de soportar grandes volúmenes de información y optimizar la toma de decisiones en un entorno altamente competitivo.

El presente informe se enmarca dentro del proyecto de creación de una plataforma diseñada para gestionar datos en un *Data Lake*, cuyo objetivo principal es facilitar la ingesta, almacenamiento y procesamiento de *market data* en tiempo real. Para lograr esto, se emplean tecnologías diseñadas para garantizar que los datos sean procesados y almacenados de manera inmediata y confiable, centrándose específicamente en el desarrollo e implementación de los módulos de ingestión y procesamiento de datos. En este contexto, se destaca la integración con herramientas clave como Apache Kafka, Apache Spark y MongoDB.

Aunque la plataforma fue diseñada específicamente para gestionar datos de mer-

cado de Binance, el desarrollo de los módulos de ingestión y procesamiento de datos permitió implementar una arquitectura modular que facilita la integración de otras fuentes de datos. Estos módulos, diseñados y ajustados para manejar *market data* en tiempo real, incrementan el potencial de la plataforma como una base adaptable para otros proyectos de Big Data, especialmente en sectores que requieren procesamiento y Análisis en Tiempo Real.

El diseño y desarrollo de esta solución no solo resuelve desafíos técnicos específicos relacionados con la ingestión y procesamiento de datos, sino que también establece una base que puede servir como referencia para proyectos futuros de análisis de datos en tiempo real. Las pruebas realizadas validan la capacidad del sistema para gestionar grandes volúmenes de información de manera escalable y eficiente, cumpliendo con los requerimientos iniciales y dejando abierta la posibilidad de futuras expansiones y mejoras.

2. Marco General

2.1. Definición del Problema

El problema principal que este proyecto busca resolver es el almacenamiento y gestión eficiente del *market data tick by tick* para una corredora de bolsa. Los datos de mercado de alta frecuencia se generan en tiempo real con cada cambio en los precios de las acciones, el volumen de transacciones y las actualizaciones en el libro de órdenes. La complejidad y la velocidad a la que se generan estos datos presentan desafíos significativos, ya que requieren una solución robusta y escalable capaz de manejar grandes volúmenes de información de manera rápida y segura.

El sistema debe ser capaz de ingerir datos en tiempo real, asegurar su almacenamiento eficiente y permitir un procesamiento y análisis. Adicionalmente, la solución debe ser escalable para adaptarse al crecimiento continuo de los datos y asegurar un almacenamiento eficiente de la información. La arquitectura del sistema debe ser lo suficientemente robusta como para manejar los desafíos técnicos que presenta la alta frecuencia de datos, incluyendo la latencia en la transmisión y la capacidad de integrarse con múltiples fuentes de datos en el futuro. Esto permitirá a la corredora acceder a datos precisos y actualizados en todo momento, optimizando la toma de decisiones y facilitando el desarrollo de estrategias de *trading* efectivas.

Estos desafíos técnicos guían el diseño de una solución integral que aborda no solo la ingestión y almacenamiento de datos, sino también su Análisis en Tiempo Real, sentando las bases para una plataforma escalable y adaptable.

2.2. Acercamiento de la Solución

Para abordar este desafío, se propone la construcción de una plataforma enfocada específicamente en el almacenamiento, integración y análisis de *market data* en tiempo real y datos históricos. La plataforma centraliza toda la información recopilada, facili-

tando su acceso y análisis, y proporciona una visión completa y precisa del mercado, lo cual resulta en una mejora en la toma de decisiones.

Como medio para implementar esta solución, se optó por un *Data Lake* que proporciona la infraestructura necesaria para soportar la integración y análisis de datos de manera eficiente. A diferencia de una base de datos tradicional, un *Data Lake* permite almacenar y gestionar grandes cantidades de datos sin comprometer el rendimiento, y brinda la flexibilidad de almacenar información en diversos formatos, desde archivos CSV y JSON, hasta texto sin formato.

Otra ventaja clave del *Data Lake* es su capacidad para realizar procesamiento en tiempo real, permitiendo a los usuarios actuar sobre los datos más recientes y mejorar la toma de decisiones en un entorno financiero dinámico y competitivo. Si bien la plataforma se diseñó inicialmente para manejar datos de Binance, su arquitectura modular permite considerar la integración de nuevas fuentes de datos en el futuro, adaptándose a contextos y requerimientos específicos.

La solución se estructuró para cumplir con objetivos técnicos clave como la escalabilidad, disponibilidad y modularidad del sistema. La elección de tecnologías de código abierto y la modularidad de la arquitectura facilitan futuras expansiones y personalizaciones, aumentando el potencial de la plataforma como base adaptable para proyectos similares en otros sectores de *Big Data* que requieren procesamiento y Análisis en Tiempo Real.

2.3. Descripción del Sistema

El sistema propuesto para gestionar *market data* de alta frecuencia está diseñado como una plataforma escalable y robusta que permite la ingesta, almacenamiento y procesamiento de grandes volúmenes de datos en tiempo real. La arquitectura del sistema se organiza en tres componentes principales, cada uno encargado de una etapa crítica en el flujo de datos:

- **Ingestión:** Los datos son extraídos de una API en tiempo real (en este caso, Binance) y transmitidos de manera continua al sistema para su procesamiento inicial.
- **Procesamiento:** Se realizan transformaciones y análisis sobre los datos capturados, generando métricas clave y estructuras organizadas.
- **Almacenamiento:** Los datos procesados y crudos se almacenan en un repositorio centralizado para su consulta y análisis posterior.

Este diseño modular asegura que cada componente sea independiente, pero altamente integrado, permitiendo la flexibilidad necesaria para incorporar nuevas funcionalidades o fuentes de datos.

2.4. Tecnologías Utilizadas

Para implementar los componentes descritos en el sistema, se seleccionaron tecnologías específicas que garantizan un manejo eficiente y escalable de los datos. A continuación, se describe cómo cada tecnología se relaciona con un componente específico:

- **Apache Kafka:** Responsable del componente de **Ingestión**, esta tecnología permite la transmisión confiable y escalable de datos en tiempo real desde la API hasta el sistema de procesamiento.
- **Apache Spark:** Parte central del componente de **Procesamiento**, Spark transforma y analiza los datos capturados en tiempo real, generando métricas clave para su interpretación.
- **MongoDB:** Implementado en el componente de **Almacenamiento**, MongoDB permite guardar tanto datos crudos como procesados de forma flexible y escalable, facilitando consultas rápidas y eficientes.

A continuación, se describen las características específicas de cada tecnología, su funcionamiento y la justificación de su uso en este proyecto:

2.4.1. Apache Kafka

Descripción: Apache Kafka es una plataforma de mensajería distribuida que permite la transmisión y almacenamiento de grandes volúmenes de datos en tiempo real. Kafka utiliza un modelo basado en productores, consumidores y *tópicos*, lo que lo hace ideal para aplicaciones que requieren alta disponibilidad y escalabilidad.

Funcionamiento Básico: Los datos son enviados por productores a un *tópico* en Kafka, donde permanecen temporalmente almacenados hasta que los consumidores los procesen. Cada mensaje cuenta con un *Offset* para mantener el orden de los eventos.

Ventajas:

- Alta escalabilidad, tolerancia a fallos y capacidad para manejar grandes volúmenes de datos en tiempo real.
- Permite la integración con múltiples sistemas de análisis y almacenamiento.

Desventajas:

- Complejidad en la configuración y mantenimiento.
- Curva de aprendizaje técnica inicial, relacionada con la configuración de tópicos y la integración con otros sistemas.

Justificación en el Proyecto: Kafka es fundamental para el proyecto porque permite manejar y distribuir *market data* de alta frecuencia en tiempo real, garantizando un flujo de datos confiable y eficiente.

2.4.2. Apache Spark

Descripción: Apache Spark es un motor de procesamiento en paralelo que permite manejar grandes volúmenes de datos distribuidos de forma eficiente. Su arquitectura

soporta análisis en tiempo real y procesamiento por lotes.

Funcionamiento Básico: Spark procesa datos mediante estructuras como RDD (Resilient Distributed Dataset) y DataFrames, aprovechando su capacidad de procesar datos en memoria para mejorar el rendimiento. Su módulo *Streaming* es esencial para análisis en tiempo real.

Ventajas:

- Procesamiento rápido gracias al uso de memoria.
- Soporte para múltiples fuentes y formatos de datos.
- Arquitectura escalable y tolerante a fallos.

Desventajas:

- Requiere un alto consumo de recursos, especialmente memoria.
- Complejidad en la configuración de clústeres distribuidos.

Justificación en el Proyecto: Spark permite procesar datos de mercado en tiempo real, generando métricas y análisis. Su rendimiento y capacidad de procesamiento paralelo son claves para cumplir los objetivos del proyecto.

2.4.3. MongoDB

Descripción: MongoDB es una base de datos NoSQL orientada a documentos que ofrece flexibilidad y escalabilidad para manejar datos no estructurados y semiestructurados.

Funcionamiento Básico: MongoDB almacena datos en formato JSON (JavaScript Object Notation), permitiendo esquemas dinámicos y consultas rápidas. Es ideal para manejar grandes volúmenes de datos de manera eficiente.

Ventajas:

- Esquema flexible, ideal para datos no estructurados.
- Escalabilidad horizontal para grandes volúmenes de datos.
- Consultas eficientes y soporte para índices avanzados.

Desventajas:

- Menor rendimiento en consultas que involucran múltiples relaciones complejas (*joins*).
- Requiere ajustes para optimizar el rendimiento en grandes volúmenes de datos.

Justificación en el Proyecto: MongoDB fue elegido por su capacidad para almacenar datos crudos y estructurados, proporcionando consultas rápidas y escalabilidad.

2.4.4. Resumen de Ventajas y Desventajas

En las secciones anteriores se analizaron las principales tecnologías utilizadas en el sistema, detallando sus características, funcionamiento y justificación. La Tabla 1 presenta un resumen de las ventajas y desventajas más importantes de estas tecnologías, proporcionando una visión clara de sus características.

Tecnología	**Ventajas**	**Desventajas**
Apache Kafka	Escalable y tolerante a fallos. Manejo eficiente de datos en tiempo real.	Complejidad en la configuración. Curva de aprendizaje por el uso de tópicos y su integración.
Apache Spark	Procesamiento rápido en memoria. Compatible con análisis en tiempo real.	Consumo elevado de memoria. Complejidad en entornos distribuidos.
MongoDB	Flexible y escalable. Consultas rápidas.	Limitado en relaciones complejas. Requiere optimización para grandes datos.

Tabla 1: *Resumen de Ventajas y Desventajas de las Tecnologías*

2.5. Estado del Arte y Técnica

El procesamiento de datos en tiempo real ha evolucionado significativamente en la última década, especialmente en industrias como la financiera, donde manejar grandes volúmenes de datos con baja latencia y alta precisión es crucial. En este contexto, las tecnologías seleccionadas para este proyecto, como Apache Kafka, Apache Spark y MongoDB, juegan un rol clave en la implementación de arquitecturas robustas y escalables. Sin embargo, también existen otras herramientas y enfoques en el ámbito de los sistemas distribuidos y de análisis de datos que han abordado desafíos similares, ofreciendo perspectivas interesantes sobre el estado del arte y la técnica en este campo.

En la industria, han surgido diversas tecnologías para gestionar datos en tiempo real, cada una con características específicas que las hacen adecuadas para diferentes casos de uso. Una de ellas es *RabbitMQ* [5], un agente de mensajes distribuido ampliamente utilizado para recopilar datos de *streaming*, provenientes de múltiples fuentes y dirigidos a diferentes destinos. Su flexibilidad y capacidad para manejar configuraciones complejas de enrutamiento lo hacen una opción atractiva para aplicaciones que requieren alta personalización. Sin embargo, en comparación con Apache Kafka, RabbitMQ presenta limitaciones en escalabilidad y manejo eficiente de grandes volúmenes de datos en tiempo real, lo que puede incrementar la complejidad operativa cuando se trabaja con cargas intensas.

Por otro lado, *Amazon Kinesis* [4] ofrece una solución completamente gestionada para la recopilación, procesamiento y análisis de datos en tiempo real. Su integración nativa con el ecosistema de AWS facilita la implementación rápida de flujos de datos, especialmente para empresas que ya operan dentro de esta infraestructura. No obstante, esta ventaja puede convertirse en una dependencia costosa y restrictiva, especialmente para organizaciones que buscan una solución más independiente o adaptable a múltiples entornos.

Otra tecnología destacada en el ámbito del procesamiento en tiempo real es *Apache Flink* [7]. Este motor es reconocido por su capacidad de procesar flujos de datos con

baja latencia y alta precisión, además de ofrecer herramientas avanzadas para la gestión de eventos complejos. Flink es ideal para casos de uso en los que se requiere un análisis continuo y detallado. Sin embargo, su implementación y mantenimiento pueden ser más exigentes en comparación con Apache Spark, lo que podría representar una barrera para equipos con menos experiencia técnica o recursos limitados.

A nivel académico, Irene Aldridge, en ”High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems” [6], subraya la importancia de la velocidad y precisión en el manejo de datos *tick by tick* para desarrollar estrategias de trading efectivas. Este enfoque resalta la necesidad de plataformas que gestionen eficientemente el flujo de datos en tiempo real, como las empleadas en este proyecto.

En el estudio ”A Big Data Lake for Multilevel Streaming Analytics” [8], Liu et al. abordan los retos asociados al manejo de datos en tiempo real mediante la implementación de Data Lakes que integren información estructurada y no estructurada. Utilizando herramientas como Hadoop Distributed File System (HDFS), los autores destacan cómo las arquitecturas basadas en Data Lakes son ideales para analizar volúmenes masivos de datos generados en tiempo real, lo que refuerza el diseño planteado en este trabajo.

Finalmente, el estudio ”Modeling Data Lakes with Data Vault: Practical Experiences, Assessment, and Lessons Learned” [2] explora cómo modelar Data Lakes utilizando técnicas como *Data Vault*. Este enfoque permite manejar la complejidad de estructuras de datos dinámicas, ofreciendo una base para el almacenamiento y análisis de información proveniente de múltiples fuentes, como ocurre en este proyecto con el *market data*.

Este marco general, junto con una evaluación de tecnologías, respalda las decisiones implementadas en el sistema. Las herramientas seleccionadas destacan por su rendimiento, escalabilidad e integración, aspectos clave para abordar los retos del análisis de datos financieros en tiempo real. A continuación, se presentan los objetivos del proyecto y cómo estas tecnologías se integran para lograrlos.

3. Objetivos

3.1. Objetivo General

Crear una plataforma escalable que permita almacenar una gran cantidad de market data, diseñada para la recuperación rápida de la información.

3.2. Objetivos Específicos

1. **Research de herramientas:** Buscar herramientas acordes a este desafío.
2. **Arquitectura del sistema:** Modelar una arquitectura que se adapte a las necesidades descritas por el cliente.
3. **Ingestión de datos:** Desarrollar un sistema capaz de recibir y almacenar datos de mercado, tick by tick, asegurando que no haya pérdida de información durante el proceso.
4. **Procesamiento de datos:** Implementar una solución en el sistema que sea capaz de procesar la información en tiempo real, que pueda manejar grandes volúmenes de datos.
5. **Almacenamiento de datos:** Implementar una solución de almacenamiento que sea eficiente y escalable, capaz de manejar grandes volúmenes de datos de alta frecuencia.

4. Diseño de la Solución

4.1. Requisitos del Sistema

4.1.1. Requisitos funcionales

RF1 **Ingestión de Datos en Tiempo Real**: El sistema debe ser capaz de recibir market data *tick by tick* de manera eficiente, cumpliendo con el requisito de ingestión continua sin pérdidas significativas.

RF2 **Almacenamiento Eficiente**: Utilizando MongoDB, el sistema debe permitir el almacenamiento de grandes volúmenes de datos de manera eficiente, asegurando que estos datos puedan ser utilizados para análisis futuros.

RF3 **Escalabilidad**: El diseño del sistema debe permitir escalar horizontal y verticalmente, asegurando su capacidad para manejar un creciente volumen de datos sin comprometer el rendimiento.

RF4 **Disponibilidad y Fiabilidad**: El sistema debe estar disponible en todo momento, con alta tolerancia a fallos para garantizar la continuidad del servicio.

RF5 **Interoperabilidad**: El sistema debe ser compatible con otras aplicaciones y sistemas utilizados por la corredora, permitiendo la integración sin problemas con otras herramientas y plataformas.

RF6 **Validación y Limpieza de Datos**: Implementar un proceso de validación y limpieza de los datos mediante Apache Spark antes de su almacenamiento. Este paso asegura la calidad y precisión de los datos, garantizando que solo información válida y limpia sea utilizada para análisis posteriores.

4.1.2. Requisitos no funcionales

RNF1 **Rendimiento:** El sistema debe ser capaz de procesar y analizar grandes volúmenes de datos en tiempo real, garantizando tiempos de respuesta adecuados incluso

bajo cargas elevadas.

RNF2 **Disponibilidad:** El sistema debe estar disponible de manera continua, lo cual es esencial para la correcta operación del *market data* en tiempo real.

RNF3 **Mantenibilidad:** Con la integración de Apache Kafka, Spark y MongoDB, el sistema debe ser modular y fácil de mantener, facilitando el monitoreo, la depuración de errores y las actualizaciones sin afectar el rendimiento.

RNF4 **Visualización y Monitorización en Tiempo Real**: Aunque no es una funcionalidad crítica, el sistema debe contar con un dashboard para la visualización y monitorización de métricas clave en tiempo real, proporcionando valor adicional a los usuarios para la toma de decisiones informadas.

4.1.3. Requisitos de interfaces

La Tabla 2 presenta los requisitos asociados a las interfaces del sistema, especificando los eventos principales, las características de cada interacción y las respuestas esperadas del sistema.

Evento	**Descripción**	**Iniciador**	**Parámetros**	**Respuesta**
Recepción de datos nuevos	El sistema recibe nueva market data para almacenamiento.	Llega un nuevo conjunto de datos al sistema.	Tipo de datos, Fuente de datos, Formato de datos	Datos almacenados y metadatos actualizados automáticamente.
Solicitud de datos	Una aplicación cliente solicita datos específicos desde el sistema.	Una petición es enviada al sistema.	ID de solicitud, Rango de fechas, Filtros específicos	Solicitud procesada y datos enviados al solicitante.

Tabla 2: *Requisitos de interfaces*

4.1.4. Requisitos de ambiente

Hardware de Desarrollo

- Servidor con capacidades de almacenamiento adecuadas para manejar grandes volúmenes de datos.
- Conexión de red estable y rápida para garantizar la transferencia eficiente de datos en el servidor.

Software de Desarrollo

- Herramientas que permiten desarrollar y administrar el sistema de almacenamiento y procesamiento.
- Sistema Operativo: Linux.
- **MongoDB 7.0.15:** Base de datos seleccionada por su flexibilidad en esquemas de datos y capacidad para manejar grandes volúmenes de datos no estructurados de forma eficiente.
- **Java 22.0.2:** Lenguaje de desarrollo elegido debido a su robustez y portabilidad entre diferentes sistemas operativos, facilitando el desarrollo y mantenimiento del back-end.
- **Apache Spark 4.0.0-preview2:** Motor de procesamiento en paralelo utilizado para realizar transformaciones de datos en tiempo real, permitiendo analizar grandes volúmenes de datos con rapidez y eficiencia.
- **Apache Kafka 3.8.0:** Plataforma de mensajería distribuida que facilita la transmisión de datos en tiempo real entre los diferentes componentes del sistema, asegurando la ingesta continua de datos sin pérdida.

- **Spring Framework 3.3.4:** Framework utilizado para gestionar la conexión y la integración de los diferentes componentes, permitiendo una arquitectura modular y escalable en el desarrollo de la aplicación.

- **Apache Zookeeper:** Servicio de coordinación esencial para la administración del clúster de Kafka, configurado en el sistema para gestionar la sincronización y la estabilidad de los nodos.

- **Docker version 26.1.4:** Plataforma de contenedores utilizada para desplegar y administrar los servicios del sistema mediante un archivo '.yml'. En el proyecto, Docker permite levantar contenedores para Zookeeper y Kafka, especificando parámetros críticos como puertos, conexiones y factores de replicación, lo que facilita la replicabilidad del entorno y la administración de servicios.

4.1.5. Perfiles de Usuario

La Tabla 3 presenta una descripción general de los diferentes grupos de usuarios, incluyendo sus características socioeconómicas, ocupacionales y etarias, junto con habilidades clave para cada perfil.

Perfil	**Socioeconómico y cultural**	**Ocupacional**	**Etario**	**Características**
Analistas de Datos	Usuarios con formación en ciencias, matemáticas o economía	Profesionales que analizan datos para identificar tendencias y patrones	25-45 años	Habilidades analíticas, capacidad de concentración, trabajo bajo presión
Traders	Usuarios con formación en finanzas y economía	Profesionales que compran y venden activos financieros	25-50 años	Toma de decisiones rápidas, alta tolerancia al riesgo, alto nivel de estrés
Desarrolladores y Administradores de Sistemas	Usuarios con formación en informática	Encargados del desarrollo y mantenimiento del sistema	22-45 años	Altas habilidades técnicas, capacidad de resolución de problemas

Tabla 3: *Perfiles de Usuario*

4.2. Arquitectura del Sistema

La arquitectura del sistema desarrollado se basa en una serie de componentes que permiten capturar, procesar y almacenar grandes volúmenes de datos en tiempo real. Para entender mejor su funcionamiento, se presentan tres diagramas que detallan la arquitectura desde diferentes niveles de abstracción: contexto general, esquema general y arquitectura específica.

4.2.1. Diagrama de contexto

El diagrama de contexto (Figura 4.1) ilustra de manera simplificada cómo interactúan las principales entidades en el flujo de datos del sistema. Los datos son generados por diversas *Fuentes de Datos* (como *Data Streams* o *Log Files*), ingresan al *Data Lake* donde se procesan y organizan, y finalmente son entregados como reportes o datos procesados a los *Consumidores* (traders, analistas financieros, algoritmos de alta frecuencia, etc.).

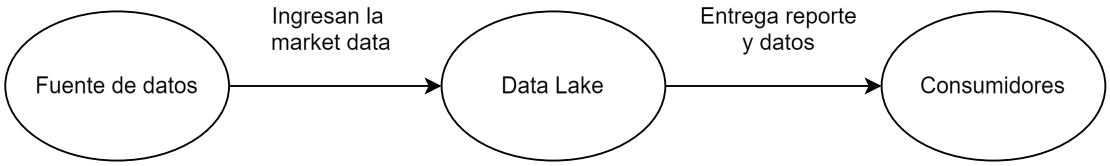


Figura 4.1: *Diagrama de contexto del sistema*

Fuente: Elaboración propia.

Este nivel de abstracción proporciona una vista general del sistema, destacando el flujo básico de datos sin entrar en detalles técnicos sobre los procesos internos.

4.2.2. Esquema general del sistema

Para entender mejor cómo se gestionan los datos en el sistema, el esquema general (Figura 4.2) detalla los principales componentes funcionales. En este diagrama se puede observar cómo las fuentes de datos son gestionadas por el módulo de *Data Ingestion*, que se encarga de recibir y preparar los datos antes de ingresarlos al *Data Lake*. El *Data Lake*, como repositorio central, permite el almacenamiento y acceso eficiente a los datos para diferentes tipos de consumidores.

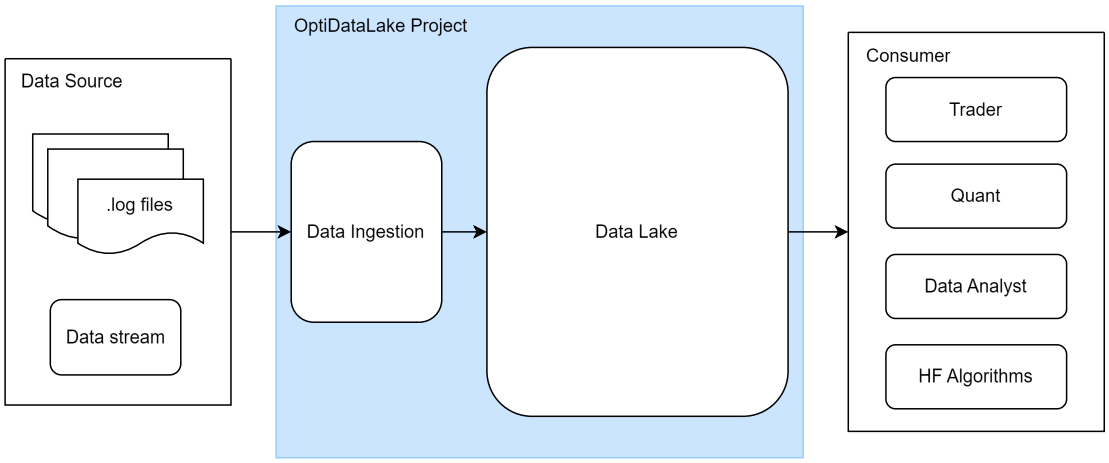


Figura 4.2: *Diagrama de alto nivel de la arquitectura del sistema*

Fuente: Elaboración propia.

Este esquema conecta las operaciones principales del sistema con los tipos de consumidores específicos, como traders, analistas cuantitativos o algoritmos, quienes utilizan los datos procesados para realizar análisis en tiempo real.

4.2.3. Diagrama de arquitectura

El último nivel de detalle lo proporciona el diagrama de arquitectura (Figura 4.3), que muestra cómo los datos fluyen dentro del sistema y cómo interactúan los diferentes módulos. Desde las fuentes de datos (*Data Streams* y *Log Files*), los datos ingresan al módulo de *Data Ingestion*, donde se procesan y estructuran antes de ser enviados al *Data Lake*. Dentro del *Data Lake*, los datos se dividen en tres áreas principales:

- **Raw/Landing:** Almacena los datos en su estado bruto, tal como fueron recibidos.
- **Structured Data:** Contiene datos organizados y listos para consultas específicas.
- **Analytics:** Almacena métricas y resultados analíticos generados por el sistema.

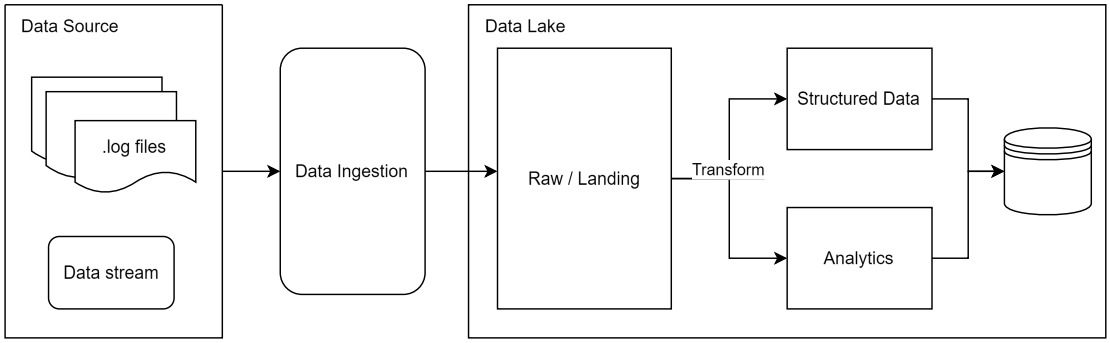


Figura 4.3: *Diagrama de alto nivel de la arquitectura del sistema*

Fuente: Elaboración propia.

Este diagrama muestra cómo los datos fluyen entre las distintas capas del sistema y cómo cada módulo cumple un rol específico en la gestión de los datos. La arquitectura modular permite que el sistema sea escalable y adaptable para futuras expansiones o integraciones de nuevas fuentes de datos.

4.2.4. Definición y Diseño de Módulos

Módulo Data Ingestion (DI)

El Módulo Data Ingestion (DI), descrito en la Tabla 4, actúa como la puerta de entrada al sistema. Se encarga de interactuar con la API de Binance para obtener datos en tiempo real, validarlos y enviarlos al módulo RD para su gestión y distribución.

Propósito	Actuar como el punto de integración con la API de Binance, extrayendo datos de mercado en tiempo real y enviándolos al módulo RD para su posterior distribución y procesamiento.
Alcance	Este módulo realiza la conexión directa con la API de Binance, extrayendo datos financieros (productos y mercados). Además, realiza una validación básica para asegurar que los datos extraídos estén en el formato esperado antes de enviarlos al módulo RD.
Dependencias	Este módulo depende exclusivamente de la API de Binance para obtener los datos y de RD para transmitirlos al resto del sistema.
Supuestos	La API de Binance estará disponible en todo momento y entregará datos válidos. Este módulo debe ser capaz de manejar un flujo de datos continuo, con un mínimo de 50 mensajes por segundo.
Restricciones	Este módulo no almacena datos de forma permanente, ya que su propósito es exclusivamente la extracción y envío. Además, se asume que los datos enviados están en formato JSON.

Tabla 4: *Descripción del módulo Data Ingestion (DI)*

Módulo Raw / Landing (RD)

El Módulo Raw / Landing (RD), descrito en la Tabla 5, funciona como intermediario en el flujo de datos del sistema. Su objetivo principal es garantizar una transmisión eficiente de los datos crudos desde el Módulo Data Ingestion (DI) hacia el Módulo Data Lake (DL) para su almacenamiento o hacia el Módulo Transform (T) para su posterior procesamiento.

Propósito	Funcionar como un intermediario para la ingesta de datos crudos provenientes del módulo Data Ingestion (DI), y distribuirlos tanto al Data Lake para almacenamiento crudo como al módulo Transform (T) para su procesamiento.
Alcance	Este módulo se encarga de recibir mensajes en tiempo real desde el DI y mantenerlos en colas temporales, permitiendo su transmisión eficiente a otros módulos del sistema, como el Data Lake o Transform.
Dependencias	Trabaja directamente con el módulo DI para recibir los datos y con los módulos Transform y Data Lake para garantizar la entrega adecuada de los mismos.
Supuestos	Los datos enviados desde DI están correctamente formateados en JSON y cumplen con los requisitos establecidos para el sistema.
Restricciones	Este módulo no almacena datos de forma permanente. Su rol es puramente transitorio y de distribución.

Tabla 5: *Descripción del módulo Raw / Landing (RD)*

Módulo Transform (T)

El Módulo Transform (T), descrito en la Tabla 6, recibe información cruda del Módulo Raw / Landing (RD) y la transforma en un formato estructurado según los requerimientos del sistema. Además, genera métricas analíticas, almacenando los resultados en el Módulo Data Lake (DL) para su consulta y análisis.

Propósito	Realizar las transformaciones de los datos crudos provenientes de RD, estructurarlos para su almacenamiento en la sección Structured Data y generar cálculos analíticos que serán almacenados en la sección Analytics del Data Lake.
Alcance	Este módulo aplica operaciones de transformación a los datos crudos para convertirlos en un formato estructurado. Además, realiza cálculos analíticos como promedios, volúmenes y precios de apertura/cierre.
Dependencias	Depende del módulo RD para recibir los datos crudos y del Data Lake para almacenar tanto los datos transformados como las analíticas calculadas.
Supuestos	Los datos crudos recibidos desde RD son válidos y contienen toda la información necesaria para las transformaciones y cálculos analíticos. Este módulo debe procesar los datos en tiempo real para garantizar un flujo continuo y eficiente del sistema.
Restricciones	Los datos transformados deben cumplir con los esquemas estructurados definidos para el sistema, y las analíticas generadas deben seguir el formato establecido para su consulta y visualización.

Tabla 6: *Descripción del módulo Transform (T)*

Módulo Structured Data (SD)

El Módulo Structured Data (SD), descrito en la Tabla 7, organiza y almacena la información procesada en un formato estructurado dentro del Módulo Data Lake (DL). Su principal objetivo es garantizar la disponibilidad de estos datos para consultas, análisis adicionales y visualizaciones específicas según los requerimientos del sistema.

Propósito	Almacenar los datos transformados en un formato estructurado dentro del Data Lake, asegurando su disponibilidad para consultas y análisis futuros.
Alcance	Recibe los datos transformados por el módulo Transform (T) y los organiza en la capa de datos estructurados del Data Lake, listos para ser consultados por otros procesos o usuarios.
Dependencias	Depende de los datos transformados por el módulo T y del Data Lake para almacenar los datos organizados.
Supuestos	Los datos transformados cumplen con los requisitos del esquema definido, y su almacenamiento en el Data Lake garantiza su disponibilidad para futuras consultas.
Restricciones	Los datos estructurados deben estar organizados y ser consistentes con el esquema definido. Una vez almacenados en el Data Lake, no pueden ser modificados.

Tabla 7: *Descripción del módulo Structured Data (SD)*

Módulo Analytics (A)

El Módulo Analytics (A), descrito en la Tabla 8, se encarga de realizar cálculos a partir de los datos provenientes del Módulo Raw / Landing (RD). Los resultados generados son almacenados en la capa de Analytics del Módulo Data Lake (DL), garantizando su disponibilidad para consultas y análisis posteriores.

Propósito	Generar métricas y analíticas avanzadas a partir de los datos, almacenándolas en la capa de Analytics del Data Lake.
Alcance	Toma datos crudos del módulo RD, calcula analíticas relevantes y almacena los resultados en la capa analítica del Data Lake, listos para ser consultados por usuarios o procesos posteriores.
Dependencias	Depende de los datos proporcionados por RD y del Data Lake para almacenar las analíticas generadas.
Supuestos	Los datos recibidos están en un estado adecuado para el cálculo de métricas, y las analíticas generadas cumplen con los requisitos definidos por el sistema.
Restricciones	Las métricas almacenadas en el Data Lake deben estar claramente indexadas y disponibles para consultas eficientes.

Tabla 8: *Descripción del módulo Analytics (A)*

Módulo Data Lake (DL)

El Módulo Data Lake (DL), descrito en la Tabla 9, funciona como el repositorio central del sistema, diseñado para almacenar y organizar datos provenientes de los diferentes módulos. En él se categorizan los datos en crudos, estructurados y analíticos, garantizando su disponibilidad para futuras consultas y análisis históricos.

Propósito	Servir como un repositorio centralizado para almacenar y organizar todos los datos del sistema, incluyendo datos crudos, transformados, estructurados y analíticos, garantizando su disponibilidad para consultas y análisis futuros.
Alcance	El módulo recibe datos de los módulos RD, SD y A, y los organiza en diferentes categorías según su nivel de procesamiento: crudos, estructurados y analíticos.
Dependencias	Este módulo depende de los datos proporcionados por RD (crudos), T (estructurados y analíticos) y los módulos SD y A para completar su función como repositorio.
Supuestos	Los datos ingresados al DL están correctamente procesados según las reglas de los módulos previos. Se garantiza que todos los datos sean almacenados de manera histórica y estén disponibles para consultas y análisis en el futuro.
Restricciones	Una vez almacenados en el DL, los datos no deben ser modificados. Además, se debe garantizar que el acceso y consulta de los datos sea eficiente incluso con volúmenes grandes de información.

Tabla 9: *Descripción del módulo Data Lake (DL)*

4.2.5. Matriz de requisitos funcionales y componentes

La Tabla 10 presenta la correspondencia entre los requisitos funcionales y los módulos que los implementan, identificando qué componente satisface cada uno de los requisitos.

	DI	**RD**	**T**	**SD**	**A**	**DL**
RF1	X					
RF2		X				X
RF3		X		X	X	X
RF4						X
RF5						X
RF6		X	X	X	X	X

Tabla 10: *Matriz de requisitos funcionales y módulos*

4.3. Estructura de Datos en el Data Lake

El modelo propuesto para almacenar y analizar datos de mercado se organiza en tres grandes categorías: **Raw Data**, **Estructurados** y **Analytics**. Esta estructura jerárquica facilita la gestión, transformación y análisis de grandes volúmenes de información financiera, asegurando que cada etapa del proceso esté optimizada para su propósito específico.

4.3.1. Datos Crudos (Raw Data)

Los datos crudos son la representación exacta de la información recibida desde la API de Binance. Esta colección contiene datos de mercado sin transformaciones ni alteraciones, preservando su integridad original para garantizar su disponibilidad en análisis históricos y procesos de validación.

Raw_data: Los datos provenientes de Binance se almacenan con las siguientes características principales:

- **eventType**: Tipo de evento registrado (e.g., *trade* para operaciones de mercado).
- **eventTime**: Marca temporal que registra el momento exacto del evento.
- **symbol**: Identificador único del activo financiero negociado, como **BTCUSDT**, **ETHUSDT** o **XRPUSDT**.
- **tradeId**: Identificador único asignado a cada transacción.
- **price**: Precio al que se realizó la transacción.
- **quantity**: Cantidad de activos financieros transaccionados.
- **tradeTime**: Hora exacta de ejecución de la transacción.

- **buyerIsMaker**: Indicador booleano que señala si el comprador es el creador de la orden (*market maker*). Este dato es esencial para análisis de profundidad de mercado.

- **marketMaker**: Indicador que determina si la transacción fue iniciada por un *market maker*, lo cual facilita el cálculo de métricas sobre la participación de proveedores de liquidez.

Los datos crudos son esenciales para procesos de transformación posteriores y cálculos analíticos avanzados.

Ejemplo de Raw_data

A continuación, se presenta un ejemplo real de cómo se almacenan los datos crudos en formato JSON:

Listing 1: *Ejemplo de Datos Crudos (Raw Data)*

```
{
    "_id": "66f60c252ec4ed50196fa5f1",
    "eventType": "trade",
    "eventTime": 1727400997811,
    "symbol": "BTCUSDT",
    "tradeId": 3861584281,
    "price": 65123.73,
    "quantity": 0.0406,
    "tradeTime": 1727400998208,
    "buyerIsMaker": true,
    "marketMaker": true
}
```

4.3.2. Estructurados

Los datos crudos pasan por un proceso de transformación y limpieza para ser almacenados en una estructura más manejable y que facilite la consulta eficiente. Estas estructuras incluyen los datos sobre los productos financieros y los mercados financieros.

Productos Financieros: Información sobre los activos financieros negociados en los mercados.

- **producto_id**: Identificador único del producto financiero.
- **nombre_producto**: Nombre del activo, por ejemplo, **Bitcoin / Tether**.

Ejemplo de Datos Estructurados: Productos Financieros

Listing 2: *Ejemplo de Datos Estructurados: Productos Financieros*

```
{
    "producto_id": "BTC/USDT",
    "nombre_producto": "Bitcoin / Tether"
}
```

Mercados Financieros: Información sobre los mercados donde se negocian los productos financieros.

- **mercado_id**: Identificador único del mercado.
- **nombre_mercado**: Nombre del mercado, por ejemplo, **Binance Exchange**.
- **tipo_mercado**: Tipo de mercado, como **Spot**.

Ejemplo de Datos Estructurados: Mercados Financieros

Listing 3: *Ejemplo de Datos Estructurados: Mercados Financieros*

```
{
    "mercado_id": "Binance",
    "nombre_mercado": "Binance Exchange",
    "tipo_mercado": "Spot"
}
```

4.3.3. Analytics (KPIs y Métricas Derivadas)

En esta categoría se almacenan los **KPIs** derivados de los datos de mercado. Estos KPIs permiten realizar análisis avanzados sobre el comportamiento de los activos financieros en los mercados. Los KPIs se calculan utilizando los datos que son extraídos de Binance y se almacenan para su análisis.

KPIs de Market Data: Indicadores clave calculados que pueden ser útiles para realizar análisis:

- **cryptocurrency**: Identificador del activo financiero, como **BTCUSDT**, **ETHUSDT** o **XRPUSDT**.
- **time_window**: Intervalo de tiempo durante el cual se recopilan y calculan los KPIs.
- **volumen_total**: Cantidad total de activos transaccionados en el intervalo de tiempo.
- **promedio_precios**: Media aritmética de los precios de cierre durante el periodo.
- **volatilidad**: Medida de la variabilidad del precio en el intervalo de tiempo.

- **movimiento_precios**: Diferencia entre el precio de apertura y el precio de cierre durante el intervalo de tiempo.

Ejemplo de Datos Analíticos (KPIs)

Listing 4: *Ejemplo de Datos Analíticos (KPIs)*


```
{
    "_id": "670fcb1fa6dcb1b4896ec5c",
    "cryptocurrency": "BTCUSDT",
    "time_window": "2024-10-16T14:18:00.000+00:00",
    "volumen_total": 53.64919000000059,
    "promedio_precios": 68222.0110529753,
    "volatilidad": 20.61907742339122,
    "movimiento_precios": 55.42999999993015
}
```


Es importante destacar que la estructura y el diseño de los datos almacenados en el sistema están pensados para garantizar su flexibilidad y adaptabilidad. Esto permite que, independientemente de si los datos son crudos, estructurados o analíticos, puedan ser ajustados, extendidos o reorganizados de acuerdo con las necesidades futuras del sistema o cambios en los requerimientos del mercado. Este enfoque asegura la escalabilidad del modelo y su capacidad para seguir siendo relevante en un entorno dinámico.

4.3.4. Esquema UML del Modelo de Datos

El diagrama UML en la Figura 4.4 muestra la relación entre las principales colecciones del Data Lake (*Raw Data*, *Estructurados* y *Analytics*), detallando cómo interactúan para cumplir los objetivos de almacenamiento y análisis del sistema:

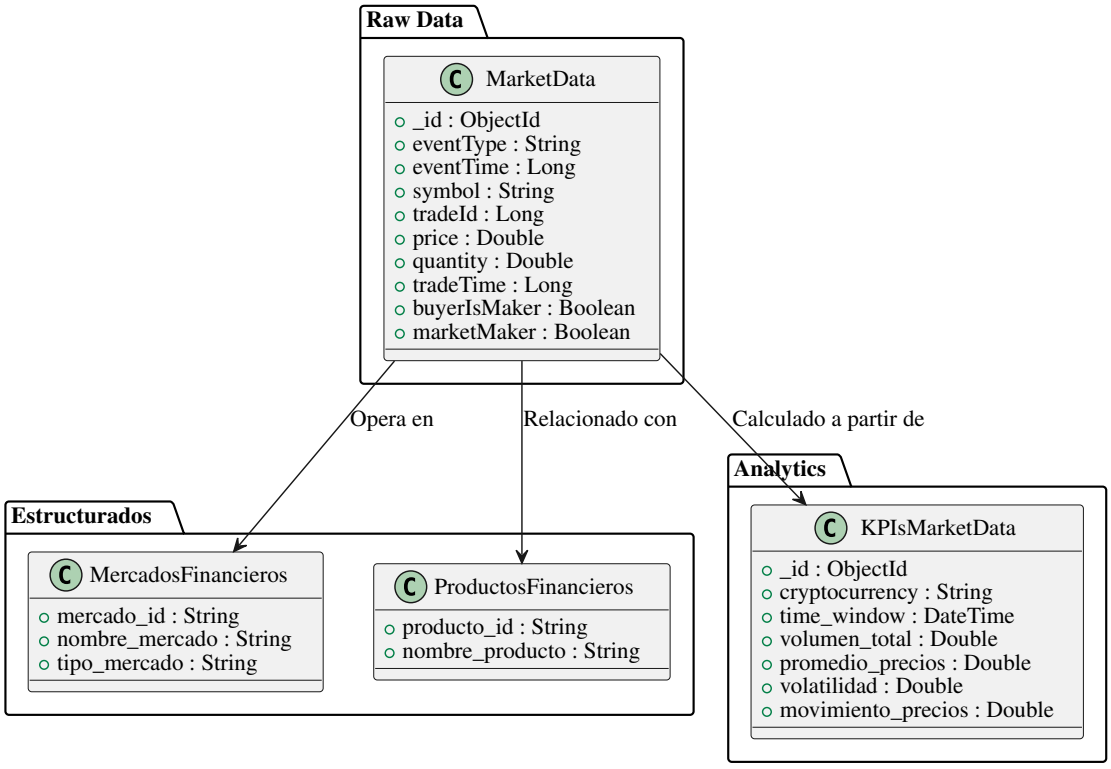


Figura 4.4: *Esquema UML del modelo de datos implementado en el Data Lake.*

Fuente: Elaboración propia.

Este esquema refleja cómo los datos crudos operan como base para las transformaciones en las colecciones estructuradas y analíticas. La clara separación entre estas etapas asegura una mayor organización y flexibilidad en el manejo de grandes volúmenes de información.

4.4. Diseño de Interfaces

4.4.1. Diagrama Entidad-Relación

El diagrama Entidad-Relación, presentado en la Figura 4.5, representa las interacciones y relaciones entre los principales componentes del sistema, detallando cómo fluye la información entre ellos.

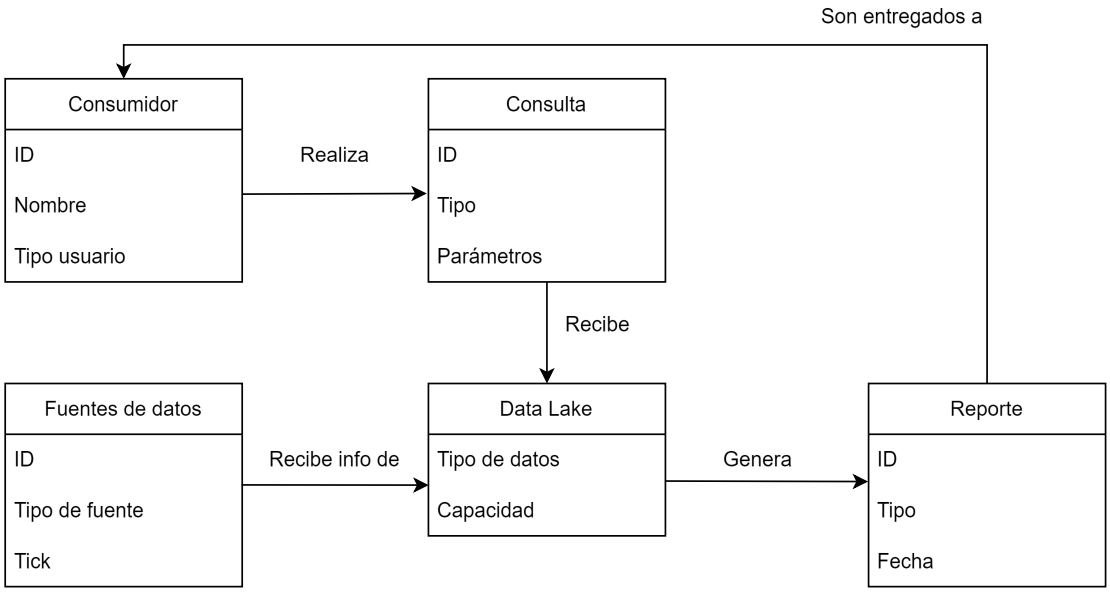


Figura 4.5: *Diagrama Entidad-Relación del sistema*

Fuente: Elaboración propia.

A continuación, se describen los principales componentes modelados en el diagrama:

1. **Consumidor**: Representa a los usuarios finales que interactúan con el sistema. Cada consumidor está identificado por un *ID*, un nombre, y un tipo de usuario, lo cual determina su rol y los permisos que tiene dentro del sistema.

2. **Consulta**: Este elemento modela las interacciones de los consumidores con el sistema. Cada consulta tiene un identificador único (*ID*), un tipo que indica su propósito, y parámetros específicos para detallar la solicitud de información.

3. **Data Lake**: Es el repositorio central del sistema donde se almacena toda la información. Este componente recibe las consultas desde los consumidores y las procesa, devolviendo la información requerida según los tipos de datos almacenados (crudos, estructurados o analíticos).

4. **Fuentes de Datos**: Representan los orígenes de la información que alimenta el

sistema. Cada fuente incluye un identificador (*ID*), el tipo de datos que genera y un *tick* asociado al tiempo de generación de los datos.

5. **Reportes**: Son el resultado de las consultas procesadas en el *Data Lake*. Cada reporte incluye un *ID*, un tipo (como analítico o resumen) y una fecha, que indica cuándo se generó.

Esta descripción resume cómo las entidades clave del sistema colaboran para ofrecer funcionalidades completas y cómo las relaciones entre las mismas permiten un flujo de información eficiente y escalable.

4.4.2. Diagrama de Flujo del Sistema

El Figura 4.6 presenta el flujo completo de datos a través del sistema, abarcando desde su captura inicial hasta su disponibilidad para los consumidores finales. Este diagrama destaca las principales etapas del proceso, como la ingesta, transformación, almacenamiento y acceso a los datos.

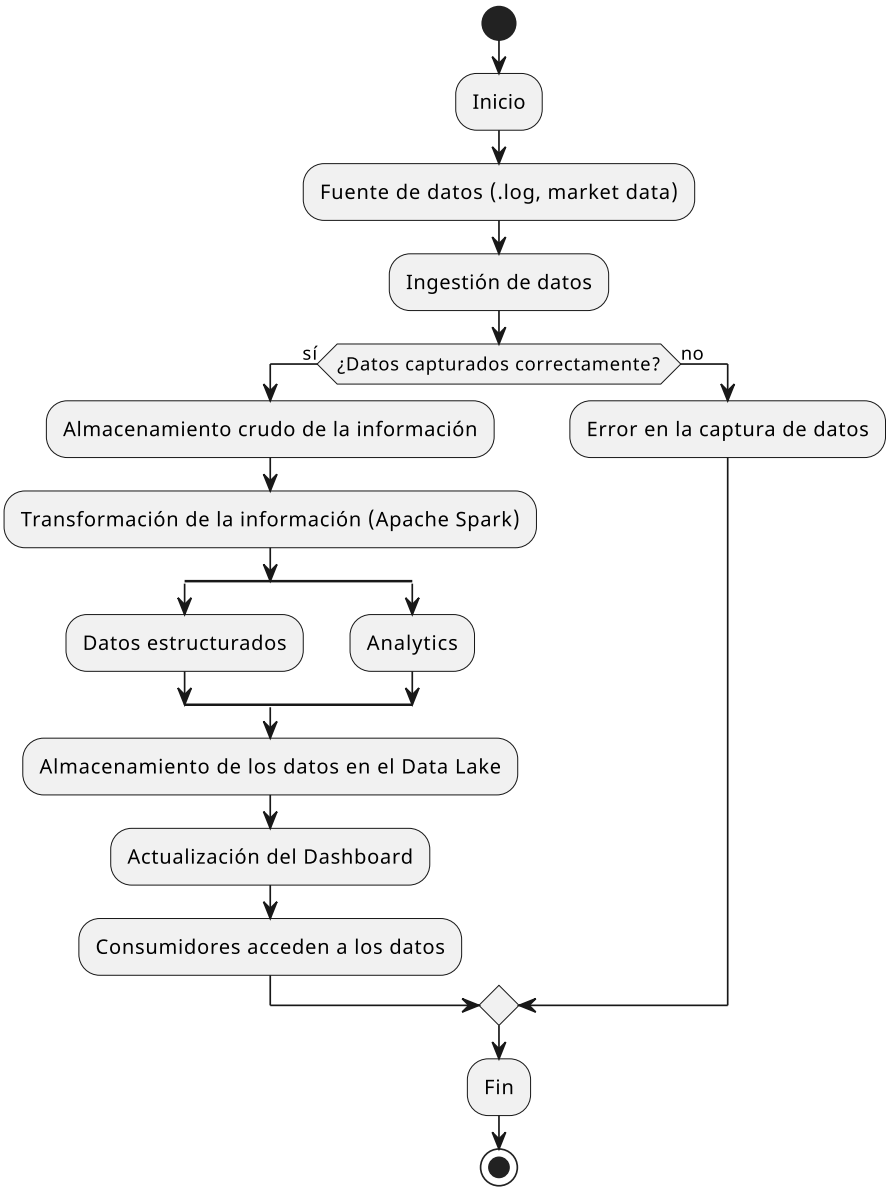


Figura 4.6: *Diagrama de flujo del sistema.*

Fuente: Elaboración propia.

A continuación, se describen las etapas representadas en el diagrama:

1. **Inicio**: Señala el inicio del proceso de manejo de datos en el sistema.

2. **Fuentes de datos**: Los datos son recibidos desde diversas fuentes, como logs del sistema o datos de mercado en tiempo real provenientes de la API de Binance.

3. **Ingestión de datos**: Los datos son capturados y preparados para ser almacenados en el sistema, asegurando que estén listos para los procesos posteriores.

4. **Almacenamiento crudo de la información**: Los datos se almacenan en su formato original (crudo) dentro de la capa correspondiente del Data Lake, preservando su estado inicial.

5. **Transformación de la información**: Los datos crudos son procesados y transformados en formatos más manejables y estructurados. Además, se generan analíticas clave para un análisis avanzado.

6. **Datos Estructurados y Analytics**: Los datos transformados son almacenados en las capas correspondientes del Data Lake:

    - **Structured Data (SD)**: Almacena datos estructurados organizados para facilitar consultas eficientes.
    - **Analytics (A)**: Contiene métricas clave y analíticas avanzadas derivadas de los datos crudos.

7. **Almacenamiento de los datos en el Data Lake**: Actúa como el repositorio central, asegurando la disponibilidad de los datos crudos, estructurados y analíticos para futuros usos.

8. **Actualización del Dashboard**: Los datos almacenados alimentan visualizaciones y métricas en un panel de control que permite el monitoreo en tiempo real.

9. **Consumidores**: Los usuarios finales o sistemas acceden a los datos almacenados en el Data Lake a través de consultas o reportes generados.

10. **Fin**: Indica el cierre del flujo de datos tras cumplir su propósito en el sistema.

Esta descripción vincula los elementos clave del diagrama con los módulos previamente definidos, lo que proporciona una visión clara y coherente del flujo de información dentro del sistema.

4.4.3. Síntesis del Diseño del Sistema

Los diagramas de flujo y entidad-relación presentados anteriormente, específicamente la Figura 4.6 y la Figura 4.5, ofrecen una descripción detallada del funcionamiento del Data Lake, abarcando desde la captura inicial de datos hasta su uso final por parte de los consumidores.

El **Diagrama de Flujo**, presentado en la Figura 4.6, ilustra el trayecto completo de los datos dentro del sistema. Este recorrido comienza con la ingestión de datos crudos desde fuentes externas, su almacenamiento inicial en formato *raw*, y la posterior transformación y almacenamiento en capas estructuradas y analíticas dentro del Data Lake. Además, incluye la integración de un *dashboard*, que proporciona visualizaciones en tiempo real basadas en las métricas clave generadas. Este enfoque garantiza un manejo eficiente de los datos, facilitando tanto su análisis como su disponibilidad inmediata para procesos críticos.

Por otro lado, el **Diagrama Entidad-Relación**, mostrado en la Figura 4.5, complementa esta visión al enfocarse en las interacciones entre los diferentes actores y componentes del sistema. Este modelo muestra cómo los consumidores interactúan con el sistema a través de consultas, las cuales desencadenan procesos que extraen, procesan y transforman los datos para generar reportes y análisis personalizados. Además, destaca cómo las fuentes de datos alimentan continuamente el Data Lake, asegurando una base de datos robusta y actualizada.

Ambos diagramas trabajan de manera conjunta para proporcionar una visión integral del Data Lake. Mientras el Diagrama de Flujo se centra en los procesos internos del manejo de datos, el Diagrama Entidad-Relación enfatiza las interacciones entre usuarios y el sistema, mostrando cómo los datos evolucionan desde su estado crudo hasta convertirse en información valiosa para la toma de decisiones.

Este diseño integral asegura que el Data Lake pueda manejar grandes volúmenes de *market data*, optimizando tanto la transformación de datos en tiempo real como su accesibilidad para los usuarios. Como resultado, los consumidores obtienen información

precisa y procesada de manera eficiente, lo cual es fundamental en un entorno de Big Data cada vez más dinámico y competitivo.

4.5. Caso de Uso

Monitoreo de Datos en Tiempo Real a través del Dashboard

El sistema desarrollado tiene múltiples aplicaciones potenciales en el análisis y gestión de datos financieros en tiempo real. Una de las principales implementaciones prácticas es el desarrollo de un **dashboard** de monitoreo en tiempo real, diseñado para facilitar la interpretación de los datos procesados. Este caso de uso, dirigido principalmente a analistas financieros y traders, representa un ejemplo práctico del alcance y versatilidad del sistema.

Actualmente, el dashboard despliega información derivada de la colección `Analytics`, incluyendo métricas como volumen total, promedio de precios y volatilidad. Sin embargo, se podría ampliar su funcionalidad para incluir datos adicionales, como información de las colecciones `Structured` o `Raw Data`, en caso de que los usuarios requieran un análisis más detallado o integral.

- **Actor Principal**: Analista financiero o trader.
- **Propósito**: Obtener una visión general en tiempo real del comportamiento del mercado financiero mediante métricas e indicadores procesados por el sistema.
- **Flujo de Trabajo**:
    1. El usuario accede al dashboard desde cualquier dispositivo con conexión al sistema, a través del Data Lake o una red compatible.
    2. Selecciona las métricas clave que desea visualizar, como volatilidad, volumen total transaccionado o movimientos de precios.

3. Observa las visualizaciones en gráficos actualizados en tiempo real, lo que le permite identificar tendencias o eventos relevantes en el mercado.

4. Utiliza los datos presentados para ajustar estrategias de trading o para realizar análisis profundos del comportamiento del mercado.

En la Figura 4.7, se presenta un ejemplo del diseño propuesto para el dashboard del sistema:

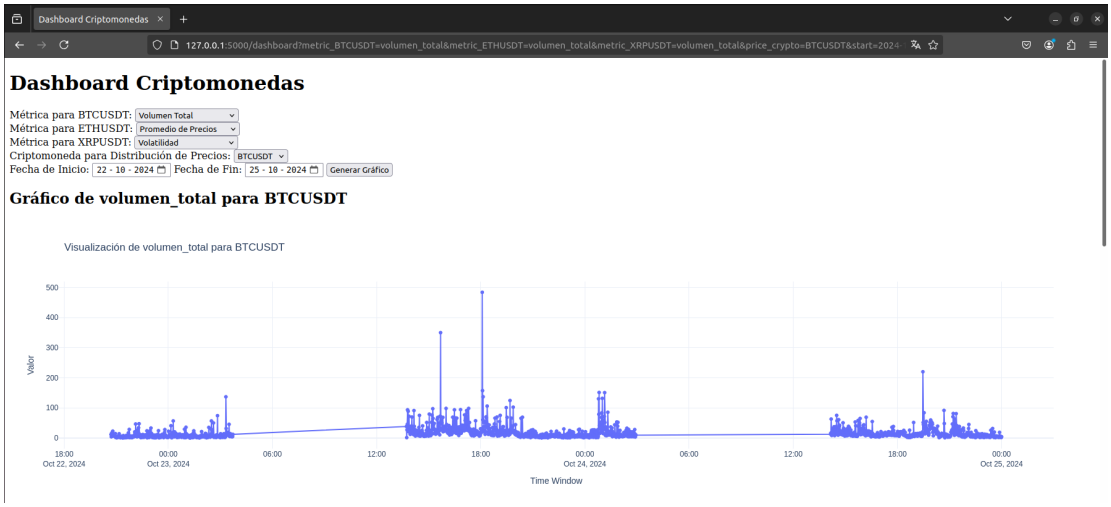


Figura 4.7: *Diseño propuesto para el dashboard de monitoreo en tiempo real.*

Fuente: Elaboración propia.

Este caso de uso destaca cómo el sistema puede ser aplicado para satisfacer necesidades específicas de monitoreo en tiempo real, proporcionando acceso a métricas clave y KPIs de manera eficiente. No obstante, este dashboard representa solo una de las múltiples aplicaciones posibles del sistema. Su diseño flexible y escalable asegura que puede ser adaptado para cubrir otros casos de uso, como la integración con algoritmos de predicción financiera, generación de reportes automatizados, o análisis avanzados requeridos por los usuarios finales.

5. Desarrollo de la Aplicación

En esta sección se describe el desarrollo técnico de los módulos del sistema, con un enfoque en la ingestión y procesamiento de datos. Asimismo, se incluye el trabajo realizado en la integración y optimización del almacenamiento, desarrollado de manera colaborativa. Estos componentes fueron diseñados, implementados y validados como parte de un proyecto global para la gestión de datos en tiempo real, asegurando una solución escalable y eficiente.

El desarrollo incluyó componentes que abarcan desde las etapas iniciales de pruebas y diseño hasta su implementación, validación y evaluación, permitiendo cumplir los objetivos establecidos como parte del sistema general.

La interacción entre los módulos principales del sistema, así como el flujo continuo de datos desde su origen hasta el almacenamiento en el Data Lake, se detalla en el **Diagrama de Componentes del Sistema** (Figura 5.1). Este diagrama proporciona una visión general de cómo las tecnologías utilizadas se integran para cumplir los objetivos del sistema.

5.1. Revisión del Sistema

El sistema desarrollado tiene como objetivo proporcionar una solución escalable y eficiente para la gestión y análisis de datos de mercado en tiempo real. Este proyecto está diseñado específicamente para el manejo de *market data* de alta frecuencia proveniente de Binance, permitiendo la ingestión, transformación y almacenamiento de grandes volúmenes de información de manera modular.

La arquitectura del sistema incorpora tecnologías como Apache Kafka, Apache Spark y MongoDB, las cuales trabajan en conjunto para garantizar un flujo de datos continuo, desde su captura inicial hasta su almacenamiento y análisis. Este enfoque modular asegura que cada componente pueda ser escalado o ajustado de manera independiente, atendiendo a los requerimientos específicos del sistema.

Si bien esta implementación se centra en una sola fuente de datos (Binance), el diseño flexible del sistema permite su posible extensión a otros entornos o sectores que requieran procesar grandes volúmenes de datos en tiempo real. De igual forma, podrían integrarse fuentes de datos adicionales para abordar escenarios más complejos, conservando su eficiencia y robustez.

5.1.1. Flujo de Trabajo para el Desarrollo del Sistema

El desarrollo del sistema se organizó en hitos principales que definieron el avance y la integración de sus componentes. A continuación, se detallan las etapas clave y las contribuciones realizadas en cada una para lograr los objetivos planteados.

Hitos Clave en el Desarrollo

- **Definición de objetivos y requisitos:** Se establecieron las necesidades principales del sistema, incluyendo el manejo de datos en tiempo real, la escalabilidad y el diseño modular.
- **Selección y evaluación de tecnologías:** Se analizaron herramientas como Apache Kafka, Apache Spark y MongoDB, seleccionándolas por su capacidad de manejar datos en tiempo real y su flexibilidad.
- **Diseño de la arquitectura del sistema:** Creación de diagramas y flujos que conectaran los componentes principales, asegurando un diseño escalable y eficiente.
- **Implementación inicial:** Desarrollo de prototipos que incluían la configuración de tópicos en Kafka y la integración con MongoDB.
- **Pruebas finales:** Validación del sistema completo, incluyendo la ingestión, transformación y almacenamiento de datos, para garantizar el cumplimiento de los requisitos funcionales.

Contribuciones del Presente Trabajo

Dentro del flujo de trabajo del sistema, se llevaron a cabo las siguientes actividades clave:

- Implementación y configuración de tópicos en Apache Kafka.
- Desarrollo de módulos para la ingestión de datos en tiempo real utilizando Spring Boot.
- Creación de scripts para la simulación de datos históricos y pruebas de estabilidad.
- Implementación de los módulos de procesamiento con Apache Spark para el cálculo de métricas clave.
- Optimización del almacenamiento en MongoDB y desarrollo del dashboard para la visualización de métricas.

En las subsecciones siguientes, se describe en detalle cómo se implementaron los componentes principales del sistema, especificando su funcionamiento y los resultados obtenidos en cada módulo.

5.1.2. Diagrama de Componentes del Sistema

El sistema se compone de varios elementos que interactúan entre sí para procesar los datos de mercado de manera eficiente. Como se muestra en la Figura 5.1, los diferentes módulos del sistema trabajan en conjunto, destacando el flujo continuo de datos desde su origen hasta su almacenamiento en el Data Lake. Aunque el sistema abarca todas las etapas desde la ingestión de datos hasta su almacenamiento, el presente trabajo pone especial énfasis en los procesos de ingestión y procesamiento, contribuyendo también en la optimización del almacenamiento.

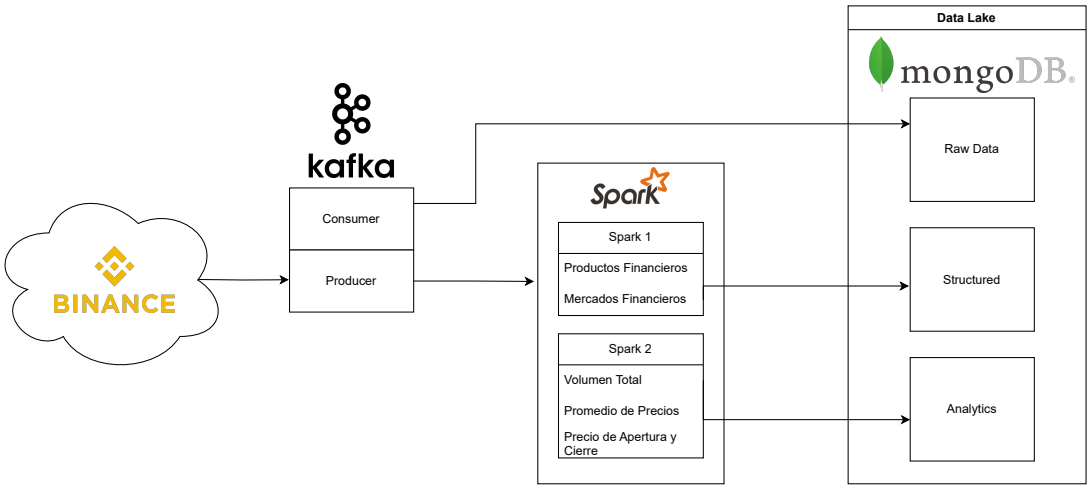


Figura 5.1: *Diagrama General de Componentes del Sistema.*

Fuente: Elaboración propia.

Componentes principales:

- **API de Binance**: Fuente de datos de mercado en tiempo real.
- **Apache Kafka**: Gestiona la ingestión y transmisión de datos mediante tópicos configurados para asegurar un flujo constante.
- **Apache Spark**: Se encarga del procesamiento y análisis de datos. Este módulo realiza la estructuración de datos financieros y el cálculo de métricas analíticas, permitiendo obtener información clave para la toma de decisiones.
- **MongoDB (Data Lake)**: Almacena los datos en sus diferentes etapas: crudos (*Raw Data*), estructurados (*Structured Data*) y analíticos (*Analytics*).

Este diagrama se incluye para mostrar cómo las tecnologías específicas, como **Apache Kafka**, **Apache Spark** y **MongoDB**, se integran en el sistema durante la etapa de implementación. A diferencia de los diagramas presentados previamente en la sección de arquitectura (Figura 4.3), este diagrama detalla la aplicación práctica de dichos conceptos, destacando las tecnologías utilizadas para realizar la ingestión, procesamiento y almacenamiento de datos.

De esta manera, se muestra cómo las decisiones de diseño arquitectónico se llevaron a la práctica, destacando las interacciones clave entre los componentes y garantizando un flujo eficiente desde la ingestión de datos hasta su análisis y almacenamiento final.

5.2. Implementación de los Componentes

5.2.1. Ingestión de Datos

La ingestión de datos en el sistema se realiza mediante la integración de la API de Binance con **Apache Kafka**. Este proceso es esencial para capturar información de *market data* en tiempo real y transmitirla de manera eficiente a través del sistema.

En el siguiente diagrama, la Figura 5.2 muestra cómo Apache Kafka gestiona el flujo de datos desde la captura inicial hasta su distribución a los módulos posteriores:

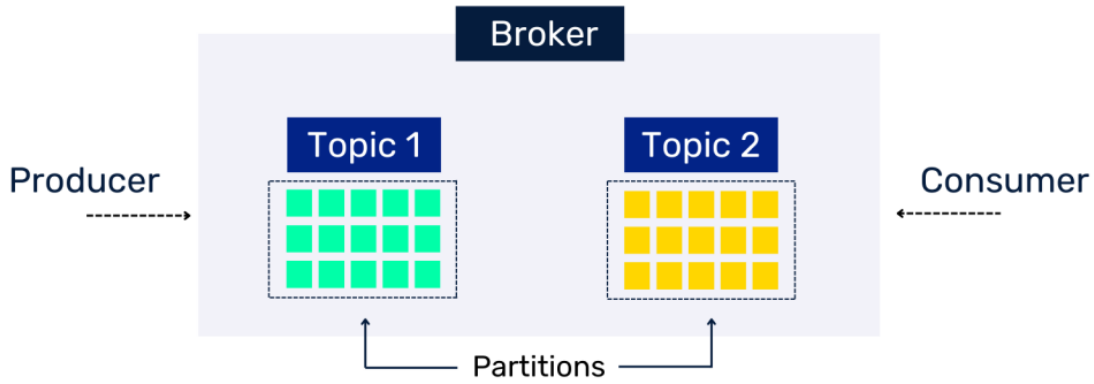


Figura 5.2: *Esquema del Funcionamiento de Apache Kafka.*

Fuente: Blog Damavis (2021), disponible en
`https://blog.damavis.com/introduccion-a-apache-kafka/`.

A continuación, se describen en detalle los componentes principales de este proceso:

Producer

El **Producer** actúa como el componente encargado de interactuar con la API de Bi-

nance. Está diseñado para recibir datos de criptomonedas en tiempo real, tales como precios, volúmenes y movimientos del mercado, y enviarlos al sistema. Las principales características del Producer incluyen:

- **Conexión Continua**: El Producer permanece conectado a la API de Binance y recibe ticks (paquetes de datos) en tiempo real, manejando múltiples eventos por segundo según la actividad del mercado.
- **Tolerancia a Fallos**: Implementa mecanismos de reconexión automática para garantizar la continuidad en caso de pérdida de conexión con la API.
- **Soporte para Múltiples Criptomonedas**: En una etapa posterior del desarrollo, se amplió el alcance del Producer para gestionar datos de múltiples criptomonedas simultáneamente. Esto incluyó la configuración de conexiones independientes para las monedas BTC/USDT, ETH/USDT y XRP/USDT, con un manejo eficiente de las limitaciones de la API de Binance (e.g., límite de mensajes por segundo).
- **Formato de Mensajes**: Los datos se envían en formato JSON, lo que facilita su procesamiento en las siguientes etapas del sistema.

Topic

Apache Kafka centraliza el flujo de datos mediante el uso de tópicos, que actúan como puntos intermedios para el almacenamiento y transmisión de mensajes. En este sistema, todos los datos recibidos desde Binance son enviados al tópico denominado `odl-ticks`. Las características clave del Topic son:

- **Centralización de Datos**: `odl-ticks` se utiliza para recibir y gestionar todos los mensajes, simplificando el procesamiento en este proyecto. Aunque un único tópico es suficiente para manejar los datos actuales, el diseño permite ampliar el sistema con tópicos adicionales en caso de integrar nuevas fuentes de datos o APIs.

- **Manejo de Mensajes Antiguos**: Un script elimina los mensajes de ejecuciones previas, asegurando que el sistema comience siempre con datos frescos.

Consumers

Los **Consumers** de Kafka son los encargados de extraer los datos almacenados en el tópico `odl-ticks` y transmitirlos a los siguientes módulos del sistema para su procesamiento. Sus principales responsabilidades son:

- **Transmisión en Tiempo Real**: Procesan los mensajes de forma continua, garantizando un flujo constante de datos hacia los componentes posteriores.
- **Validación de Datos**: Verifican que los datos extraídos sean completos y libres de errores antes de enviarlos al módulo de procesamiento.

5.2.2. Contribuciones en el Módulo de Ingestión de Datos

El módulo de ingestión de datos fue diseñado para capturar información de *market data* en tiempo real desde la API de Binance utilizando **Apache Kafka**. A continuación, se describen las etapas clave en el desarrollo e implementación del módulo de ingestión de datos:

Primera Etapa: Configuración Inicial con Docker

Se creó un archivo `docker-compose.yml` para levantar un clúster local con Apache Kafka y Zookeeper. En esta etapa, se configuró el tópico `odl`, verificando la conexión entre el productor y el consumidor.

Segunda Etapa: Desarrollo del Productor y Consumidor Básico

Se implementaron scripts en Java para crear un productor y un consumidor iniciales. En esta fase, se validó la transmisión de mensajes en formato `clave:valor` y se garantizó que los consumidores pudieran procesar mensajes secuenciales.

Tercera Etapa: Integración con la API de Binance

Se integró el framework Spring Boot para conectar la API de Binance con Apache Kafka. Esto permitió:

- Publicar datos de criptomonedas en tiempo real en el tópico `odl-ticks`.
- Asegurar la transmisión eficiente de los datos capturados desde la API hacia el sistema, manejando múltiples eventos por segundo.

Cuarta Etapa: Manejo de Mensajes Antiguos en el Consumer

Se desarrolló un script para gestionar los mensajes almacenados en el tópico `odl-ticks`. Este script elimina los mensajes antiguos de ejecuciones previas, asegurando que el sistema siempre procese datos frescos. Esto fue clave para mantener la eficiencia y la consistencia en los datos procesados por los consumidores.

Quinta Etapa: Ampliación para Múltiples Criptomonedas

Como parte de la evolución del sistema, se amplió el módulo de ingestión para soportar datos de múltiples criptomonedas en tiempo real. En esta etapa, se configuraron conexiones independientes para BTC/USDT, ETH/USDT y XRP/USDT, superando las limitaciones de la API de Binance, que restringe la cantidad de mensajes por segundo. Se implementaron las siguientes mejoras:

- **Reconexión Automática**: Se añadió un mecanismo de reconexión en caso de que se alcanzaran los límites de mensajes por segundo, asegurando la continuidad del flujo de datos.
- **Optimización de Rendimiento**: Se ajustó el Producer para manejar las tres criptomonedas simultáneamente, sin comprometer la latencia o la calidad de los datos.
- **Validación de Datos**: Se verificó que los datos de las criptomonedas fueran consistentes y libres de errores antes de ser enviados al tópico `odl-ticks`.

Estas etapas reflejan el trabajo realizado por el autor para garantizar que el módulo de ingestión funcione correctamente y sea capaz de manejar grandes volúmenes de datos de manera eficiente y confiable.

5.2.3. Procesamiento de Datos

El procesamiento de los datos recibidos se lleva a cabo utilizando **Apache Spark**, que permite realizar transformaciones y análisis en tiempo real de los datos ingresados desde **Kafka**. Spark, gracias a su arquitectura distribuida y su compatibilidad con *Spark Streaming*, ofrece una solución eficiente y escalable para manejar grandes volúmenes de datos en tiempo real. A continuación, se detallan los componentes internos de Apache Spark:

Componentes de Apache Spark

Apache Spark opera con una arquitectura basada en **Master Node** y **Worker Nodes**, lo cual permite realizar procesamiento distribuido y paralelo. La Figura 5.3 muestra la arquitectura general de Spark y cómo se gestiona el flujo de datos en este sistema:

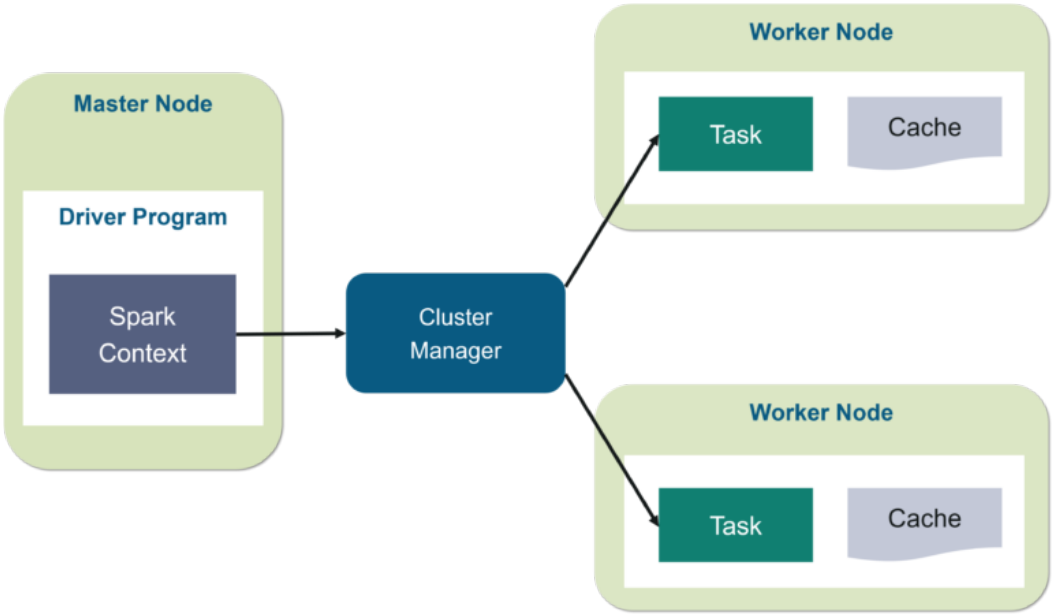


Figura 5.3: *Arquitectura de Apache Spark utilizada para el procesamiento de datos.*

Fuente: Edureka Blog (2024), disponible en `https://www.edureka.co/blog/spark-architecture/`.

A continuación, se detallan los componentes principales:

- **Master Node**: Es el nodo principal que ejecuta el *Driver Program* y crea el contexto de Spark (*Spark Context*). Este componente divide el trabajo y lo distribuye entre los **Worker Nodes**.
- **Cluster Manager**: Coordina los recursos del clúster, asignando tareas a los nodos disponibles. En este proyecto, se utilizó el **Modo Local**, una configuración simplificada que simula un entorno distribuido en un solo equipo, ideal para pruebas y desarrollo.
- **Worker Nodes**: Ejecutan las tareas asignadas por el **Master Node**. Los datos son procesados en paralelo, y cada nodo puede almacenar temporalmente los datos en memoria (*Cache*) para acelerar el acceso y mejorar el rendimiento.

Con esta arquitectura, Apache Spark proporciona un procesamiento distribuido efi-

ciente. A continuación, se destacan algunas de sus principales ventajas en el contexto de este sistema.

Ventajas del Procesamiento en Spark

El uso de Apache Spark proporciona las siguientes ventajas:

- **Escalabilidad**: Permite procesar datos en paralelo utilizando clústeres distribuidos.
- **Tolerancia a Fallos**: Spark gestiona automáticamente la reanudación del procesamiento mediante mecanismos como la persistencia de datos y los reintentos automáticos de micro-batches.
- **Compatibilidad**: Integra de manera nativa con **Kafka**, **MongoDB** y otras herramientas.
- **Flexibilidad**: Soporte para múltiples transformaciones y cálculos analíticos sobre grandes volúmenes de datos.

Flujo de Procesamiento

El procesamiento de datos en este sistema sigue una secuencia que permite transformar datos crudos provenientes de **Kafka** en información valiosa para análisis financiero. Cada etapa está diseñada para asegurar la calidad de los datos, optimizar su organización y calcular métricas que ofrezcan una visión integral del mercado. A continuación, se describe el flujo de procesamiento:

1. **Lectura desde Kafka**: Spark actúa como consumidor del tópico `odl-ticks`, recibiendo los datos en tiempo real.
2. **Transformaciones en Spark**: Los datos crudos se estructuran y transforman en formatos más manejables (*DataFrames*), aplicando filtros y transformaciones necesarias para asegurar la calidad de la información.

3. **Cálculo y Organización de Datos**:
    - **Estructuración de Datos Financieros**: Spark organiza datos relacionados con:
        - **Productos Financieros**: Información sobre productos como criptomonedas, que incluye identificadores únicos y nombres descriptivos de los productos.
        - **Mercados Financieros**: Información sobre mercados financieros, que incluye su identificador, nombre y tipo de mercado (e.g., spot, futuros).
    - **Cálculo de Métricas Analíticas**: Spark calcula métricas clave como:
        - **Volumen Total**: Suma del volumen de transacciones en la ventana de tiempo definida.
        - **Promedio de Precios**: Cálculo del precio promedio de las transacciones realizadas en el intervalo.
        - **Volatilidad**: Medida de la dispersión de los precios en la ventana de tiempo, que representa la variabilidad del mercado.
        - **Movimiento de Precios**: Diferencia entre el precio de apertura y el precio de cierre durante el intervalo de tiempo.
4. **Escritura en MongoDB**: Los resultados procesados son almacenados en dos colecciones principales:
    - `Analytics`: Contiene las métricas analíticas calculadas, organizadas por ventanas de tiempo específicas.
    - `Structured`: Almacena los datos estructurados relacionados con mercados y productos financieros.

5.2.4. Contribuciones en el Módulo de Procesamiento de Datos

El módulo de procesamiento fue diseñado para estructurar y analizar datos en tiempo real provenientes del sistema de ingestión. A continuación, se describen las etapas clave desarrolladas en el módulo de procesamiento de datos:

Primera Etapa: Configuración Inicial de Spark

Se configuró Spark en **Modo Local** para realizar pruebas iniciales y validar su integración con Apache Kafka. En esta etapa, se logró:

- Consumir datos directamente desde el tópico `odl-ticks`.
- Configurar el contexto de Spark Streaming para procesar los datos en micro-batches de 1 segundo.

Segunda Etapa: Limpieza y Preparación de Datos Crudos

En esta etapa, se desarrollaron transformaciones iniciales para limpiar los datos provenientes de **Kafka** y dejarlos listos para su procesamiento posterior. Las actividades clave incluyeron:

- **Limpieza de Datos**: Eliminación de valores nulos, duplicados y otros inconsistentes en los datos crudos.
- **Formateo Estandarizado**: Conversión de datos crudos a un formato estructurado utilizando *DataFrames* en Apache Spark, facilitando el procesamiento eficiente.

Tercera Etapa: Cálculo de Métricas Analíticas

Se implementaron transformaciones para calcular métricas analíticas clave, utilizando ventanas de tiempo configurables (1 minuto por omisión). Las métricas calculadas son volumen total, precio promedio, volatilidad y movimiento de precios.

Cuarta Etapa: Escritura en MongoDB

Se amplió la funcionalidad del flujo de procesamiento en Spark para incluir la escritura de los datos procesados en la colección `Analytics` de MongoDB. Esto incluyó:

- Validación de la conexión entre Spark y MongoDB.
- Configuración para realizar escrituras incrementales en tiempo real.

Estas etapas permitieron que el módulo de procesamiento cumpliera con los requisitos del sistema, garantizando un flujo eficiente de datos desde la ingestión hasta el almacenamiento final.

5.2.5. Almacenamiento de Datos

El almacenamiento de datos en el sistema se realiza utilizando **MongoDB**, una base de datos NoSQL orientada a documentos que permite manejar grandes volúmenes de datos de manera flexible y eficiente. Los datos se organizan en diferentes colecciones según su nivel de procesamiento, facilitando su consulta y análisis.

La configuración inicial y la creación de las colecciones `Raw Data`, `Structured` y `Analytics` se realizó utilizando **MongoDB Compass**, lo que permitió una gestión rápida y visual de las estructuras requeridas. En este sistema, los datos llegan a MongoDB desde dos fuentes principales:

- **Apache Kafka**: Proporciona datos crudos (*Raw Data*) directamente desde la API de Binance.
- **Apache Spark**: Envía datos procesados y organizados en dos categorías: *Structured Data* y métricas analíticas (*Analytics*).

A continuación, se describen las principales colecciones utilizadas:

Raw Data

La colección **Raw Data** almacena los datos crudos directamente recibidos de la API de

Binance a través de Kafka. Estos datos se guardan sin modificaciones para garantizar su disponibilidad en análisis históricos o en caso de requerir reprocesamientos. Los principales campos almacenados son:

- **eventType**: Tipo de evento, como *trade*.
- **eventTime**: Marca temporal del evento.
- **symbol**: Activo financiero (e.g., **BTCUSDT**).
- **price** y **quantity**: Precio y cantidad del activo negociado.
- **tradeTime**: Hora de la transacción.

Structured Data

La colección **Structured Data** almacena los datos que han sido transformados y organizados mediante Spark. Estos incluyen información relacionada con:

- **Productos Financieros**: Detalles de los activos financieros, como *producto_id* y *nombre_producto*.
- **Mercados Financieros**: Información sobre los mercados donde se realizan las transacciones, incluyendo *mercado_id*, *nombre_mercado* y *tipo_mercado*.

Aunque este módulo no fue desarrollado como parte del enfoque de este trabajo, se incluye para proporcionar una visión completa del sistema y explicar cómo los datos son organizados en esta etapa.

Analytics

La colección **Analytics** almacena las métricas calculadas por Spark Streaming, derivadas de los datos de mercado procesados en tiempo real. Estas métricas incluyen:

- **volumen_total**: Cantidad total de activos negociados.

- **promedio_precios**: Media aritmética de los precios dentro de una ventana de tiempo.
- **volatilidad**: Variación relativa de los precios en el periodo.
- **movimiento_precios**: Diferencia entre el primer y último precio registrado en la ventana.

Optimización y Consultas

Para garantizar un acceso eficiente a los datos almacenados, se aplicaron estrategias como:

- **Indexación**: Los campos más consultados, como `symbol`, `eventTime` y `time_window`, fueron indexados para acelerar las búsquedas.
- **Estructuración por Colecciones**: Los datos fueron segmentados en colecciones específicas para reducir la carga en las consultas y optimizar la organización del almacenamiento.

Para los fines de este proyecto, MongoDB actúa como un *Data Lake*, centralizando los datos provenientes de diferentes etapas del sistema en un solo repositorio. Si bien MongoDB cumple este rol en el contexto actual, un *Data Lake* completo podría requerir la integración de herramientas adicionales para ingesta, almacenamiento y procesamiento distribuido, especialmente en casos donde se manejen mayores volúmenes de datos o múltiples fuentes de información.

5.2.6. Contribuciones en el Módulo de Almacenamiento de Datos

En el desarrollo del módulo de almacenamiento, las principales aportaciones realizadas incluyeron:

Creación de Colecciones y Organización Inicial

Se colaboró en el diseño y creación de las colecciones `Raw Data` y `Analytics`, utilizando **MongoDB Compass** para definir las estructuras necesarias según los requerimientos del sistema.

Optimización de Consultas con Indexación

Se implementaron índices en campos como `symbol` y `eventTime`, para mejorar el rendimiento en las consultas. Estas optimizaciones permiten una mayor eficiencia en el acceso a los datos almacenados.

Con este trabajo, el almacenamiento en **MongoDB** cumplió con los objetivos de eficiencia y flexibilidad necesarios para el sistema. Además, el diseño y la implementación del almacenamiento establecieron una base para las etapas de validación y pruebas del sistema, donde se verificará su rendimiento, estabilidad y cumplimiento de los objetivos definidos.

5.3. Validación y Pruebas del Sistema

En esta sección se presentan las pruebas realizadas para validar el correcto funcionamiento de los componentes desarrollados e implementados directamente. Estas pruebas abarcan desde la ingestión de datos hasta su procesamiento y almacenamiento en las colecciones `Raw Data` y `Analytics`. Aunque otros módulos, como `Structured`, forman parte del sistema, las validaciones documentadas aquí se limitan a los desarrollos mencionados.

5.3.1. Ingestión de Datos desde la API de Binance

Se verificó que los datos provenientes de Binance fueran enviados correctamente al *Topic* `odl-ticks` de Apache Kafka mediante un **Producer**. A continuación, la Figura 5.4 presenta la evidencia de los mensajes enviados:

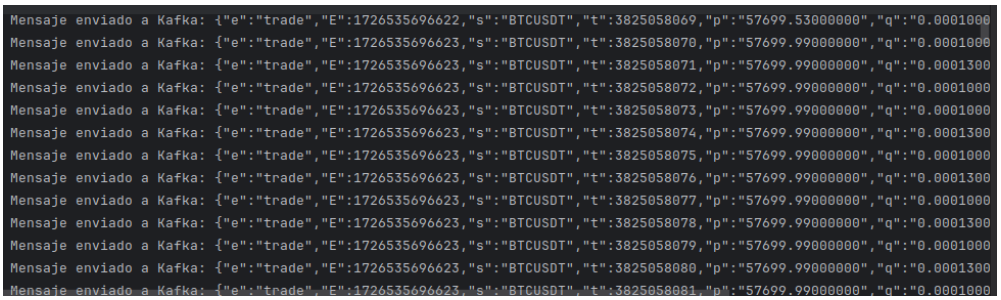


Figura 5.4: *Producer enviando datos desde Binance hacia Apache Kafka.*

Fuente: Elaboración propia.

Posteriormente, se validó que un **Consumer** configurado en el sistema recibiera correctamente estos mensajes para ser procesados. La Figura 5.5 muestra cómo los mensajes son consumidos y enviados a MongoDB:

Figura 5.5: *Consumer recibiendo y procesando datos desde Apache Kafka.*

Fuente: Elaboración propia.

5.3.2. Procesamiento de Datos con Apache Spark

En esta etapa, se validó que Apache Spark pudiera consumir los datos crudos desde Kafka, procesarlos y calcular métricas derivadas como volumen total y volatilidad. La Figura 5.6 muestra la ejecución de este procesamiento en tiempo real:

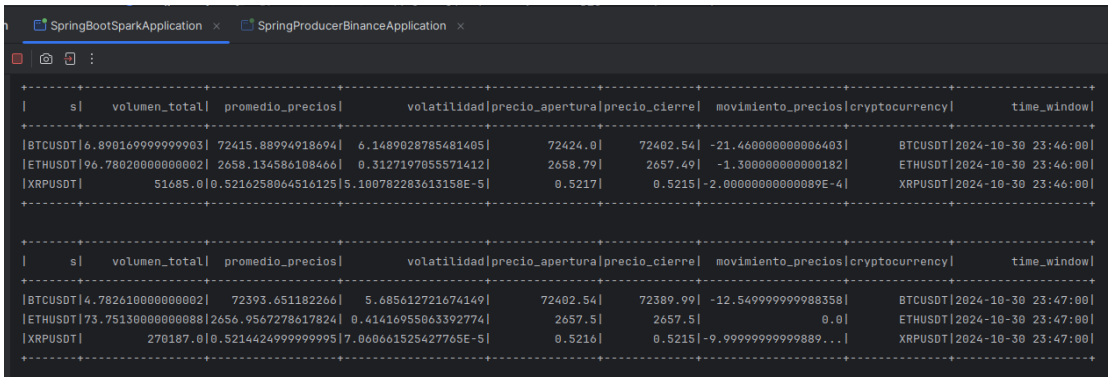


Figura 5.6: *Cálculo de métricas mediante Apache Spark.*

Fuente: Elaboración propia.

5.3.3. Almacenamiento de Datos en MongoDB

Se verificó que los datos, tanto crudos como procesados, fueran almacenados correctamente en MongoDB. La Figura 5.7 muestra cómo la colección `Raw_Data` contiene los datos crudos provenientes de Binance:

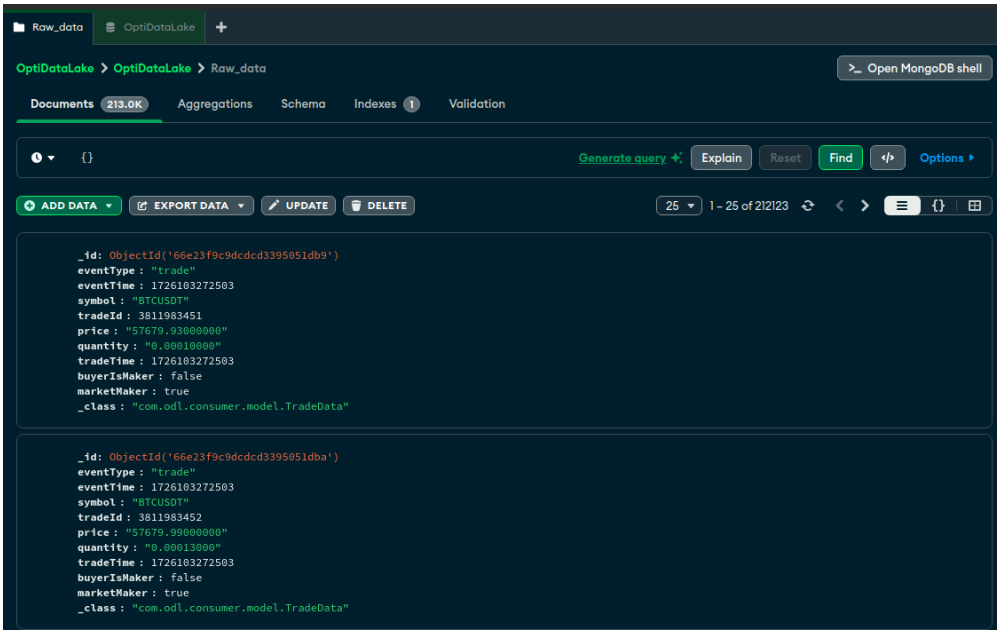


Figura 5.7: *Datos crudos almacenados en la colección* `Raw_Data` *de MongoDB.*

Fuente: Elaboración propia.

Por otro lado, los resultados procesados por Spark, incluyendo las métricas calculadas, se almacenaron en la colección `Analytics`, como se observa en la Figura 5.8:

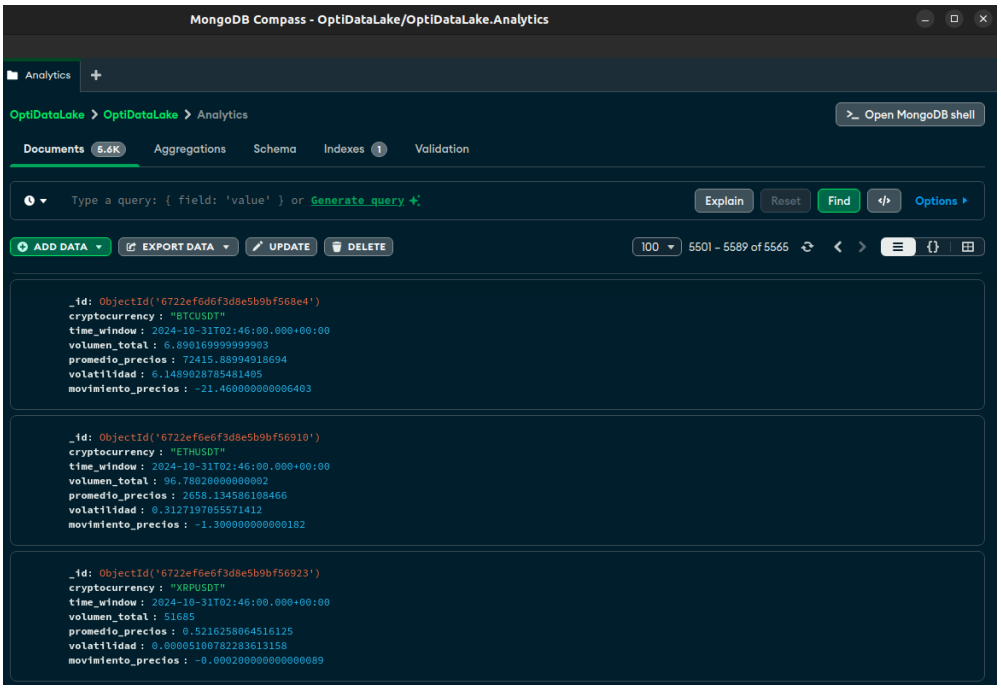


Figura 5.8: *KPIs calculados y almacenados en la colección* `Analytics` *de MongoDB.*

Fuente: Elaboración propia.

5.3.4. Resumen de Contribuciones en la Validación y Pruebas del Sistema

Las validaciones realizadas se enfocaron en los módulos desarrollados, garantizando su correcto funcionamiento y documentando su rendimiento. Entre las principales actividades llevadas a cabo se incluyen:

- Verificación de la correcta ingestión de datos desde la API de Binance hacia el *Topic* `odl-ticks` en Apache Kafka, asegurando la integridad y continuidad de los datos capturados.
- Validación del procesamiento de datos en tiempo real mediante Apache Spark, incluyendo el cálculo de métricas analíticas clave y la escritura de resultados en

la colección `Analytics` de MongoDB.

- Pruebas de almacenamiento en las colecciones `Raw Data` y `Analytics`, comprobando que los datos se organizaran y almacenaran según las especificaciones del sistema.

Estas validaciones proporcionaron un respaldo técnico sólido al sistema, destacando la funcionalidad y estabilidad de los módulos implementados.

5.4. Pruebas de Rendimiento

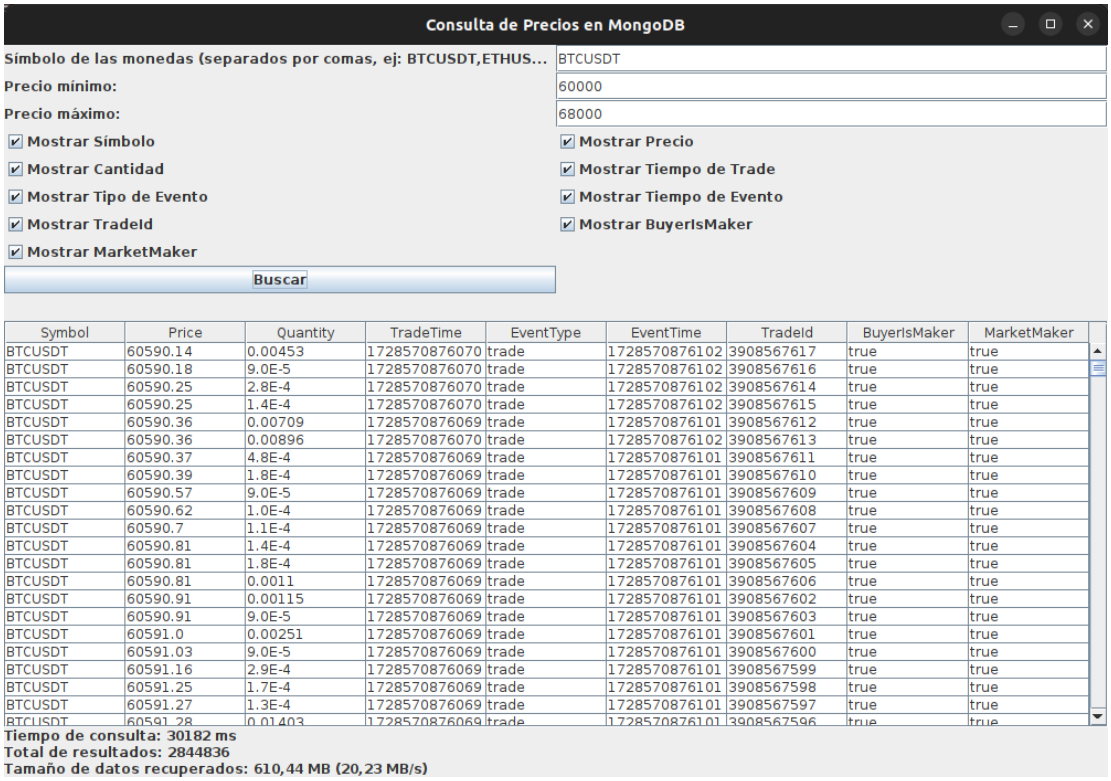
En esta sección se evalúa el rendimiento del sistema, considerando su capacidad para manejar grandes volúmenes de datos en tiempo real y su eficiencia en la recuperación y almacenamiento de información. Las pruebas se centran en el rendimiento general, la ingestión de datos y la estabilidad del sistema.

5.4.1. Pruebas de Rendimiento General

Estas pruebas midieron el tiempo de respuesta y el ancho de banda utilizado durante consultas realizadas sobre los datos almacenados en *Raw Data*. Para esto, se diseñaron consultas específicas que evaluaron la eficiencia del sistema bajo diferentes escenarios de carga.

Interfaz de Consultas

Se utilizó una interfaz para ejecutar y explorar consultas en MongoDB, diseñada para facilitar la validación del rendimiento del sistema. Como se muestra en la Figura 5.9, esta herramienta permitió recuperar datos en intervalos específicos, asegurando su funcionalidad bajo diferentes volúmenes de datos:



Symbol	Price	Quantity	TradeTime	EventType	EventTime	TradeId	BuyerIsMaker	MarketMaker
BTCUSDT	60590.14	0.00453	1728570876070	trade	1728570876102	3908567617	true	true
BTCUSDT	60590.18	9.0E-5	1728570876070	trade	1728570876102	3908567616	true	true
BTCUSDT	60590.25	2.8E-4	1728570876070	trade	1728570876102	3908567614	true	true
BTCUSDT	60590.25	1.4E-4	1728570876070	trade	1728570876102	3908567615	true	true
BTCUSDT	60590.36	0.00709	1728570876069	trade	1728570876101	3908567612	true	true
BTCUSDT	60590.36	0.00896	1728570876070	trade	1728570876102	3908567613	true	true
BTCUSDT	60590.37	4.8E-4	1728570876069	trade	1728570876101	3908567611	true	true
BTCUSDT	60590.39	1.8E-4	1728570876069	trade	1728570876101	3908567610	true	true
BTCUSDT	60590.57	9.0E-5	1728570876069	trade	1728570876101	3908567609	true	true
BTCUSDT	60590.62	1.0E-4	1728570876069	trade	1728570876101	3908567608	true	true
BTCUSDT	60590.7	1.1E-4	1728570876069	trade	1728570876101	3908567607	true	true
BTCUSDT	60590.81	1.4E-4	1728570876069	trade	1728570876101	3908567604	true	true
BTCUSDT	60590.81	1.8E-4	1728570876069	trade	1728570876101	3908567605	true	true
BTCUSDT	60590.81	0.0011	1728570876069	trade	1728570876101	3908567606	true	true
BTCUSDT	60590.91	0.00115	1728570876069	trade	1728570876101	3908567602	true	true
BTCUSDT	60590.91	9.0E-5	1728570876069	trade	1728570876101	3908567603	true	true
BTCUSDT	60591.0	0.00251	1728570876069	trade	1728570876101	3908567601	true	true
BTCUSDT	60591.03	9.0E-5	1728570876069	trade	1728570876101	3908567600	true	true
BTCUSDT	60591.16	2.9E-4	1728570876069	trade	1728570876101	3908567599	true	true
BTCUSDT	60591.25	1.7E-4	1728570876069	trade	1728570876101	3908567598	true	true
BTCUSDT	60591.27	1.3E-4	1728570876069	trade	1728570876101	3908567597	true	true
BTCUSDT	60591.28	0.01403	1728570876069	trade	1728570876101	3908567596	true	true

Figura 5.9: *Interfaz para consultas sobre los datos almacenados en MongoDB.*

Fuente: Elaboración propia.

Resultados

El gráfico de la Figura 5.10 presenta la relación entre el volumen total de datos recuperados y el ancho de banda utilizado durante las pruebas:

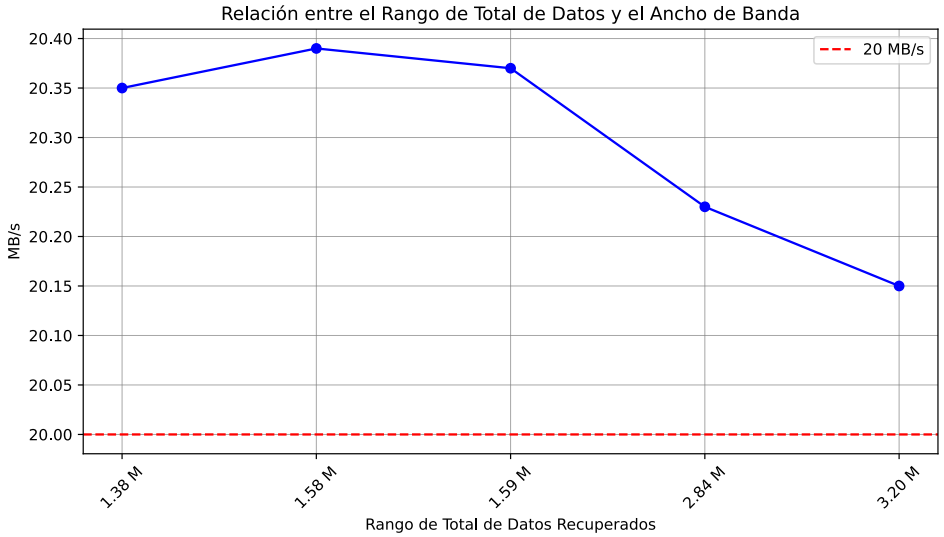


Figura 5.10: *Relación entre el rango de datos recuperados y el ancho de banda.*

Fuente: Elaboración propia.

Como se observa en la Figura 5.10, el sistema mantiene un rendimiento estable, logrando cumplir con el objetivo planteado de alcanzar un rendimiento mínimo de ingestión de datos de 20 MB/s. Esto valida la eficiencia del sistema al procesar grandes volúmenes de datos sin comprometer su rendimiento, satisfaciendo así las expectativas del proyecto.

5.4.2. Pruebas de Ingestión de Datos por Minuto

Se evaluó la cantidad de mensajes ingresados al sistema por minuto en la colección *Raw Data*. Este análisis, realizado por miembros del equipo, permitió validar la capacidad del sistema para manejar un flujo continuo de datos sin interrupciones.

Resultados

El gráfico de la Figura 5.11 muestra la cantidad de datos ingresados por minuto, evidenciando un flujo constante de información:

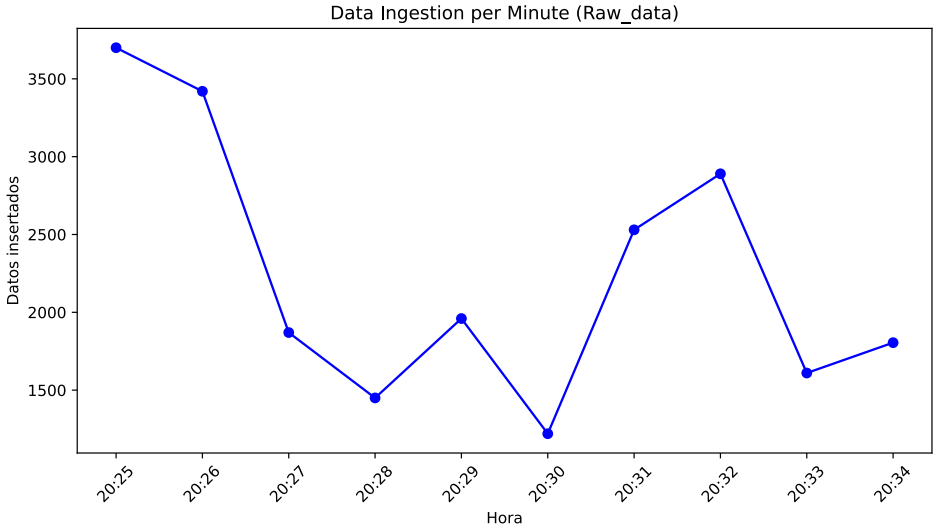


Figura 5.11: *Cantidad de datos ingresados por minuto en la colección Raw Data.*

Fuente: Elaboración propia.

Como se observa en la Figura 5.11, el sistema mantiene un flujo constante de ingestión, incluso durante picos de actividad. Esto demuestra que el diseño e implementación del sistema pueden manejar la carga esperada sin pérdidas de datos. Además, este resultado valida la eficacia del script desarrollado para monitorear el ingreso de datos en tiempo real.

5.4.3. Evaluación del Sistema Completo

En general, el sistema demostró un rendimiento adecuado tanto en la ingestión como en la recuperación de datos, cumpliendo con los objetivos establecidos para el manejo de datos en tiempo real. La integración entre **Apache Kafka**, **Apache Spark** y **MongoDB** proporcionó un flujo de datos eficiente y sin interrupciones, garantizando una operación confiable y estable.

El principal objetivo del proyecto era alcanzar un rendimiento mínimo de ingestión de datos de 20 MB/s. Este objetivo fue cumplido y validado mediante pruebas controladas, diseñadas para simular escenarios representativos del flujo de datos esperado. Aunque no se trabajó con cargas reales, los resultados obtenidos demuestran que el sistema puede manejar grandes volúmenes de datos en tiempo real, cumpliendo con los estándares y expectativas establecidos para el proyecto.

5.4.4. Contribuciones en las Pruebas de Rendimiento

En este apartado, se destacan las principales actividades realizadas en colaboración y de forma independiente:

- Participación en la configuración y validación de la interfaz de consultas para MongoDB, diseñando consultas dinámicas y evaluando el rendimiento en la colección *Raw Data*.
- Realización de pruebas y análisis de recuperación de datos en intervalos específicos para validar el throughput del sistema, asegurando que cumpliera con el objetivo de superar los 20 MB/s. Estas pruebas se llevaron a cabo utilizando datos almacenados previamente, partiendo de un rango base de precios de BTC de 60000 y expandiéndolo progresivamente en intervalos de 2000 hasta alcanzar 70000. Este enfoque permitió evaluar el rendimiento del sistema en escenarios de consulta con diferentes volúmenes de datos.
- Participación en la revisión de los resultados y análisis preliminar de las métricas relacionadas con la ingestión de datos por minuto en *Raw Data*, una vez desarrolladas por el equipo.

Estas actividades garantizaron que el sistema cumpliera con los requerimientos de rendimiento establecidos y proporcionaron una base sólida para su validación final.

5.5. Conclusión

Este proyecto logró el desarrollo de un sistema integral para la gestión de datos de mercado en tiempo real, empleando tecnologías como **Apache Kafka**, **Apache Spark** y **MongoDB**. El sistema fue diseñado y validado para cumplir con los objetivos propuestos, destacando su capacidad para:

- La implementación y validación del módulo de ingestión de datos, asegurando una transmisión continua y confiable desde la API de Binance hacia el sistema.
- El diseño y desarrollo de transformaciones en **Apache Spark** para procesar datos crudos y calcular métricas clave, como volumen total, promedio de precios y volatilidad.
- La configuración y optimización de consultas en **MongoDB**, mejorando la eficiencia en la recuperación y almacenamiento de grandes volúmenes de datos.
- La creación del **dashboard**, que proporciona un monitoreo visual y en tiempo real de métricas analíticas clave para los usuarios del sistema.

Estas contribuciones permitieron validar que el sistema cumple con los objetivos planteados, entre ellos la capacidad para ingerir datos de alta frecuencia, procesarlos en tiempo real y garantizar un rendimiento mínimo de 20 MB/s. Además, el desarrollo del proyecto permitió implementar soluciones funcionales en cada etapa del sistema, desde la ingestión de datos hasta su almacenamiento, asegurando un flujo continuo y estable. Esto no solo estableció una base sólida para el análisis actual, sino que también abre oportunidades para futuras optimizaciones en aspectos como la escalabilidad, el rendimiento y la incorporación de nuevas fuentes de datos.

5.6. Trabajo Futuro

Aunque el sistema alcanzó los objetivos planteados, se identificaron varias áreas de mejora que podrían ser abordadas en el futuro:

- **Ampliación de Funcionalidades**: Incorporar nuevas fuentes de datos y métricas personalizadas en el **dashboard**, adaptándose a las necesidades de diferentes usuarios y escenarios.

- **Optimización del Sistema**: Implementar estrategias avanzadas de optimización en **Apache Spark** y MongoDB para mejorar el rendimiento en entornos de mayor complejidad.

- **Pruebas en Entornos Reales**: Validar el sistema bajo condiciones reales de mercado financiero para evaluar su desempeño y robustez en escenarios de producción.

- **Escalabilidad y Disponibilidad**: Migrar la solución a plataformas en la nube, como *AWS* o *Google Cloud*, para asegurar una mayor escalabilidad y resiliencia frente a picos de carga.

- **Análisis Predictivo**: Integrar herramientas de aprendizaje automático para agregar funcionalidades de predicción y modelado, enriqueciendo el análisis proporcionado por el sistema.

Estas líneas de trabajo futuro no solo mejorarían la capacidad del sistema, sino que también expandirían su aplicabilidad a nuevos desafíos y contextos, reforzando su relevancia en la gestión de datos en tiempo real.

Referencias

[1] Edureka Blog. Spark architecture - understanding components & applications. https://www.edureka.co/blog/spark-architecture/, 2024. Fecha de acceso: 3 de diciembre de 2024.

[2] Giebler Corinna, Gröger Christoph, Hoos Eva, Schwarz Holger, and Mitschang Bernhard. Modeling data lakes with data vault: Practical experiences, assessment, and lessons learned. In *Advances in Conceptual Modeling*. Springer, 2019.

[3] Blog Damavis. Introducción a apache kafka. https://blog.damavis.com/introduccion-a-apache-kafka/, 2021. Fecha de acceso: 3 de diciembre de 2024.

[4] AWS Documentation. Amazon kinesis data streams. https://aws.amazon.com/kinesis/data-streams/, 2023. Fecha de acceso: 20 de diciembre de 2024.

[5] AWS Documentation. What is rabbitmq? https://aws.amazon.com/es/compare/the-difference-between-rabbitmq-and-kafka/, 2023. Fecha de acceso: 10 de diciembre de 2024.

[6] Aldridge Irene. *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. Wiley Trading, 2013.

[7] Rubén Alejandro Jaime. Estudio comparativo entre apache flink y apache spark. https://sedici.unlp.edu.ar/handle/10915/126780, 2021. Fecha de acceso: 11 de diciembre de 2024.

[8] Liu Ruoran, Isah Haruna, and Zulkernine Farhana. A big data lake for multilevel streaming analytics. https://ieeexplore.ieee.org/document/9245460, 2020. Fecha de acceso: 19 de noviembre de 2024.

Anexos

Repositorio del Proyecto

El código fuente del sistema desarrollado, junto con los scripts y configuraciones utilizados, se encuentra disponible en el siguiente repositorio de GitHub:

- `https://github.com/ArielBravoP/OptiDataLake-PMM`

Si bien el repositorio no contiene una guía exhaustiva para la replicación completa del sistema, incluye los elementos principales que pueden ser utilizados como base para su estudio o ampliación.