Trabajo Final: Procesamiento de Grandes Volúmenes de Datos

Maestría de Ciencia de Datos - UNAJ

Raúl Burgos Mauro Cejas Marcovecchio

2026-02-15

Introducción

En el contexto del análisis de grandes volúmenes de datos, las arquitecturas distribuidas juegan un rol fundamental para el procesamiento eficiente y escalable de información. Tecnologías como Apache Spark y Apache Kafka se han convertido en estándares para el procesamiento batch, streaming y la ingestión de datos en tiempo real.

El objetivo de este trabajo es diseñar e implementar un clúster virtualizado que permita simular una infraestructura orientada al análisis de datos del mercado financiero. Para ello, se utilizó Docker como tecnología de virtualización liviana, desplegando un clúster compuesto por tres nodos que integran Apache Spark, Apache Kafka y Apache Zookeeper.

El caso de uso se inspira en un concurso académico realizado en 2022, donde se propuso el análisis en tiempo real de datos de mercado. Dado que la API original ya no se encuentra disponible, se optó por simular la ingestión de datos utilizando Kafka como sistema de mensajería distribuido.

Desarrollo

El sistema diseñado consta de tres niveles distintos y cinco nodos:

Arquitectura de infraestructura

La implementación se basó en la tecnología de contenedores Docker, utilizando Docker Compose como orquestador para definir una red aislada denominada *bigdata-network*. Esta red permite la comunicación entre nodos, garantizando un entorno reproducible y escalable.

- **Zookeeper**: servicio de coordinación para gestionar el estado del broker, las cuotas y, lo más importante, la elección de líderes para las particiones de los temas (topics).

```yaml
zookeeper:
  image: confluentinc/cp-zookeeper:7.0.1
  container_name: zookeeper
  ports:
    - "2181:2181"
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  networks:
    - bigdata-network
```

- Kafka: encargado de la ingestión y distribución de eventos que simulan datos del mercado financiero, que describiremos más adelante. Fue configurado con múltiples **listeners** para diferenciar el tráfico interno del clúster y el tráfico externo de los productores de datos.

```
kafka:
  image: confluentinc/cp-kafka:7.0.1
  container_name: kafka
  depends_on:
    - zookeeper
  ports:
    - "9092:9092"
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
    KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka:29092,EXTERNAL://localhost:9092
    KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  networks:
    - bigdata-network
```

- Spark: por un lado, tendremos un master, que no procesa datos directamente sino que su función tareas a los workers y gestionar el ciclo de vida de las aplicaciones Spark. Por el otro, habrán dos nodos workers, que recibirán las divisiones del trabajo realizadas por el master, llamadas tasks, para ejecutarlas y devolver los resultados, garantizando la escalabilidad horizontal.

La virtualización mediante Docker permitió cumplir con el requerimiento sin necesidad de máquinas virtuales completas, reduciendo el consumo de recursos y simplificando el despliegue.

Configuración del clúster

El despliegue se realizó mediante un archivo docker-compose.yml, donde se definieron: - Imágenes oficiales de Spark, Kafka y Zookeeper - Variables de entorno necesarias para la correcta inicialización - Puertos expuestos para acceso a servicios (Spark UI y Kafka) - Configuración específica de Kafka para operar con un único broker, ajustando los factores de replicación internos

Un aspecto clave del desarrollo fue la correcta configuración de "advertised.listeners" y de los factores de replicación internos de Kafka, necesarios para evitar errores de liderazgo en entornos de un solo nodo.

Flujo de información

En primer lugar, la generación de datos está a cargo de un productor desarrollado en Python que actúa como nuestro "simulador de mercado", emitiendo eventos constantes (Ticker, Timestamp, Precio). Cada evento contiene el identificador del activo, un timestamp y un valor de precio, permitiendo evaluar el comportamiento del sistema ante un flujo continuo de datos en tiempo real.

En segundo lugar, el transporte se realiza mediante Apache Kafka que recibe estos eventos en el tópico "market-data", asegurando que ningún dato se pierda en el camino.

Por último, el procesamiento es realizado con Spark Structured Streaming que se encarga de la parte "pesada": lee el flujo, limpia el formato CSV y realiza cálculos matemáticos.

Cabe destacar que todo este ecosistema (Zookeeper, Kafka y Spark) conviva en una red aislada y lista para desplegarse en cualquier entorno se debe a la orquestación Mediante Docker Compose.

Estrategia de análisis

En este punto es importante resaltar que no nos limitamos solo a leer datos, sino que también los organizamos. Para ello implementamos ventanas temporales de 30 segundos, lo que nos permite observar el comportamiento de activos en bloques de tiempo manejables.

Además, añadimos un Watermark de 1 minuto (vital en sistemas reales) que nos permite ser tolerantes y esperar por datos que puedan llegar con un ligero retraso debido a la red, sin detener el procesamiento global.

Pruebas

En primer lugar se verificó el correcto funcionamiento de cada componente:

- Spark Master: Acceso exitoso a la interfaz web de Spark (http://localhost:8080), confirmando que el servicio se encontraba operativo.
- Kafka Broker: Ejecución de comandos administrativos para listar y crear tópicos.
- Zookeeper: Verificación indirecta a través del correcto funcionamiento de Kafka.

Como prueba funcional del sistema se implementó un caso de uso simple de ingestión de datos. Se enviaron mensajes simulando datos de mercado financiero y, mediante un consumidor Kafka, se verificó la recepción exitosa de los mensajes producidos, confirmando el correcto funcionamiento del flujo productor–broker–consumidor. Este procedimiento valida que el clúster es capaz de manejar eventos en tiempo real, cumpliendo con el objetivo de simular una arquitectura de streaming de datos.

Durante las pruebas el sistema demostró una sincronía impecable. Validamos que el flujo end-to-end funcionara correctamente, desde la creación del mensaje en Python hasta la actualización del promedio de precios en la consola de Spark. Asimismo, al agrupar los datos, logramos una visión clara del mercado en ventanas específicas, obteniendo promedios de precios precisos para cada Ticker de forma casi instantánea.

Conclusiones

En este trabajo se logró implementar exitosamente un clúster Big Data compuesto por tres nodos virtualizados utilizando Docker. La infraestructura desplegada permite simular un entorno realista de ingestión y procesamiento de datos, integrando Apache Spark y Apache Kafka. Podríamos destacar como punto fuerte del proyecto:

- El despliegue de un clúster distribuido con tecnologías ampliamente utilizadas en la industria.
- La implementación de un caso de uso funcional de ingestión de datos en tiempo real.
- La resolución de problemas reales asociados a la configuración de Kafka en entornos Docker, demostrando comprensión del funcionamiento de la plataforma.

La solución desarrollada cumple con los requerimientos planteados y sienta las bases para la extensión hacia escenarios de procesamiento más complejos.

En síntesis, hemos logrado construir una base sólida y reproducible. El uso de Docker ha sido un acierto para la portabilidad, aunque aprendimos que la gestión de scripts mediante volúmenes es un punto clave para la persistencia.

Trabajo a futuro

Este pipeline es solo el comienzo. Para llevar esta solución al siguiente nivel, nuestras metas a corto plazo son:

- Ampliación: Crear otras métricas financieras que permitan un análisis más amplio para la toma de decisiones.
- Visualización: Conectar los resultados a un dashboard para ver las curvas de precios en tiempo real.
- Almacenamiento: Persistir las métricas en una base de datos No SQL (como Cassandra) para análisis históricos.

Referencias

Apache Software Foundation. (n.d.). *Kafka structured streaming programming guide*. Apache Spark Documentation. Retrieved https://spark.apache.org/docs/latest/streaming/index.html

Apache Software Foundation. (2026). *Apache kafka documentation*. https://kafka.apache.org/documentation.

Calavaro, Russo, Cardelini. (2022). *Realtime analysis of market data leveraging apache flink*. https://doi.org/10.1145/3524860.3539650.

Confluent Inc. (2026). *Confluent kafka configuration guide*. https://docs.confluent.io.

Docker Inc. (2026). *Docker compose documentation*. https://docs.docker.com/compose.

Xie, Dervieux, Riederer. (2026). *R markdown cookbook*. https://yihui.org/rmarkdown-cookbook/.