

# Procesamiento de Grandes Volúmenes de Datos

Katherine Rodríguez Rodríguez      Reinaldo Cánovas Gamón

**Proyecto 2:** Análisis Histórico de Datos Financieros

## Contents

<b>1</b>	<b>Introducción</b>	<b>1</b>
<b>2</b>	<b>Ingesta de Datos</b>	<b>2</b>
2.1	API Alphavantage . . . . .	2
2.2	Apache Kafka . . . . .	2
2.3	Captura de datos . . . . .	3
<b>3</b>	<b>Almacenamiento de Datos</b>	<b>4</b>
3.1	The Hadoop Distributed File System (HDFS) . . . . .	4
<b>4</b>	<b>Referencias</b>	<b>5</b>

## 1 Introducción

El análisis de datos es una disciplina que ha experimentado un notable crecimiento en todos los campos, así como en organizaciones y empresas, debido a la necesidad de contar con herramientas que analicen datos y que estos sirvan para la toma de decisiones eficaces y eficientes.

Los datos han crecido de manera exponencial en las últimas décadas, lo que ha generado una serie de problemas, como el acceso a esa cantidad de datos, su extracción y almacenamiento. Por lo tanto, se introduce aquí la definición de *Big Data*.

*Big data* se refiere a conjuntos de datos extremadamente grandes y complejos que no pueden gestionarse ni analizarse fácilmente con las herramientas tradicionales de procesamiento de datos.

Para abordar estos desafíos, se emplean diversas herramientas avanzadas como Hadoop y Apache Kafka. Hadoop es conocido por su capacidad de distribuir y procesar grandes conjuntos de datos a través de clústeres, facilitando el almacenamiento y la gestión eficiente de datos de gran volumen. Por otro lado, Apache Kafka es una plataforma de procesamiento de flujos que permite la

transmisión y gestión de grandes volúmenes de datos de manera rápida y eficiente, garantizando que la información fluya de forma continua y sin interrupciones.

Este trabajo no solo pretende abordar los desafíos asociados con la gestión de grandes volúmenes de datos, sino también demostrar su aplicación práctica a través de un análisis histórico de datos financieros, ofreciendo a analistas e inversores una comprensión profunda de los mercados financieros y respaldando estrategias de inversión informadas y efectivas.

## 2 Ingesta de Datos

### 2.1 API Alphavantage

Para realizar la extracción de datos históricos sobre finanzas, se utiliza la API Alpha Vantage. Esta API devuelve la cadena de opciones histórica completa para un símbolo específico en una fecha determinada, abarcando más de 15 años de historia. También proporciona la volatilidad implícita (IV) y los griegos comunes (por ejemplo, delta, gamma, theta, vega y rho). Las cadenas de opciones se ordenan por fechas de vencimiento en orden cronológico. Dentro de la misma fecha de vencimiento, los contratos se organizan por precios de ejercicio de menor a mayor.

.La volatilidad implícita es una medida de la expectativa del mercado derivada de su precio.

.Los "griegos" son medidas que describen cómo se espera que los precios de las opciones respondan a diferentes factores del mercado:

**Delta:** Mide la sensibilidad del precio de la opción a cambios en el precio del activo subyacente.

**Gamma:** Mide la tasa de cambio de Delta sobre el precio del activo subyacente.

**Theta:** Representa la tasa de cambio del precio de la opción con respecto al tiempo.

**Vega:** Mide la sensibilidad del precio de la opción a cambios en la volatilidad del activo subyacente.

**Rho:** Mide la sensibilidad del precio de la opción a cambios en la tasa de interés.

### 2.2 Apache Kafka

Kafka es un sistema distribuido que consta de servidores y clientes que se comunican a través de un protocolo de red TCP de alto rendimiento. Puede implementarse en hardware sin sistema operativo, máquinas virtuales y contenedores, tanto en entornos locales como en la nube.

**Servidores:** Kafka se ejecuta como un clúster de uno o más servidores que pueden abarcar varios centros de datos o regiones en la nube. Algunos de estos servidores forman la capa de almacenamiento, conocidos como brokers. Otros servidores ejecutan Kafka Connect para importar y exportar continuamente datos como flujos de eventos, integrando Kafka con sistemas existentes, como bases de datos relacionales y otros clústeres de Kafka. Para permitir la implementación de casos de uso críticos, un clúster de Kafka es altamente escalable y tolerante a fallos; si alguno de sus servidores falla, los demás asumirán su carga de trabajo para garantizar operaciones continuas sin pérdida de datos.

**Clientes:** Los clientes permiten escribir aplicaciones distribuidas y microservicios que leen, escriben y procesan flujos de eventos en paralelo, a gran escala y de manera tolerante a fallos, incluso ante problemas de red o fallos en las máquinas.

Los **productores** son las aplicaciones cliente que publican (escriben) eventos en Kafka, mientras que los **consumidores** son aquellos que se suscriben (leen y procesan) estos eventos. En Kafka, los productores y los consumidores están totalmente desacoplados y son agnósticos entre sí, lo cual es un elemento clave en el diseño para lograr la alta escalabilidad por la que Kafka es conocido. Por ejemplo, los productores nunca tienen que esperar a los consumidores. Kafka ofrece varias garantías, como la capacidad de procesar eventos exactamente una vez.[1]

## 2.3 Captura de datos

Se implementa el entorno de Kafka utilizando Docker y Docker Compose, lo que nos permite configurar y gestionar de manera rápida y escalable los contenedores.

Para la extracción de datos históricos, se optó por el método de procesamiento en lotes. Este enfoque resultó más conveniente, ya que los datos históricos suelen ser grandes y estáticos. Se configura la consola y el diseño de Rich para una visualización mejorada. También se configuran el servidor Kafka y el productor de Kafka, que enviará mensajes a un tema (topic) específico. Los mensajes serán serializados en formato JSON.

```
1         from kafka import KafkaProducer
2             import json
3         from rich.console import Console
4         from rich.layout import Layout
5
6         console = Console()
7         layout = Layout()
```

```

8         KAFKA_TOPIC = 'ibm_options'
9         KAFKA_SERVER = 'localhost:9092'
10        producer = KafkaProducer(
11            bootstrap_servers=[KAFKA_SERVER],
12            value_serializer=lambda x: json.dumps(x).encode('
13                utf-8')
        )

```

El consumer se configura para escuchar en el tema "ibm-options" en el servidor de Kafka especificado. A medida que recibe mensajes, los deserializa de JSON a un diccionario de Python.

```

1        consumer = KafkaConsumer(
2            KAFKA_TOPIC,
3            bootstrap_servers=KAFKA_SERVER,
4            value_deserializer=lambda x: json.loads(x.decode('utf-8')),
5            auto_offset_reset='earliest',
6            enable_auto_commit=True
7        )

```

## 3 Almacenamiento de Datos

### 3.1 The Hadoop Distributed File Systemp (HDFS)

El sistema de archivos distribuidos de Hadoop (HDFS) es un sistema de archivos diseñado para gestionar grandes conjuntos de datos que pueden ejecutarse en hardware básico. HDFS es el sistema de almacenamiento de datos más popular para Hadoop y permite escalar un único clúster de Apache Hadoop a cientos e incluso miles de nodos. Dado que gestiona de forma eficiente grandes volúmenes de datos con un alto rendimiento, HDFS se puede utilizar como canalización de datos y es ideal para soportar análisis de datos complejos.[2]

En este trabajo, hay una función llamada save-data que se encarga de guardar los mensajes recibidos del Kafka Consumer en un sistema de archivos distribuido HDFS o localmente. Si el cliente HDFS (hdfs-client) está configurado, intenta guardar el archivo en el directorio HDFS especificado (HDFS-DIR). Se utiliza with hdfs-client.write para escribir los datos en un archivo en HDFS. Si el archivo se guarda correctamente, se imprime un mensaje de éxito en la consola y se devuelve el tipo de almacenamiento (HDFS) junto con el nombre del archivo (ver Figura 1).

```

1 def save_data(data, timestamp):
2     """Guarda los datos en HDFS o localmente"""
3     filename = f"ibm_options_{timestamp}.json"
4
5     if hdfs_client:
6         try:
7             file_path = f"{HDFS_DIR}/{filename}"
8             with hdfs_client.write(file_path, encoding=
9                 'utf-8') as writer:
10                 json.dump(data, writer)

```

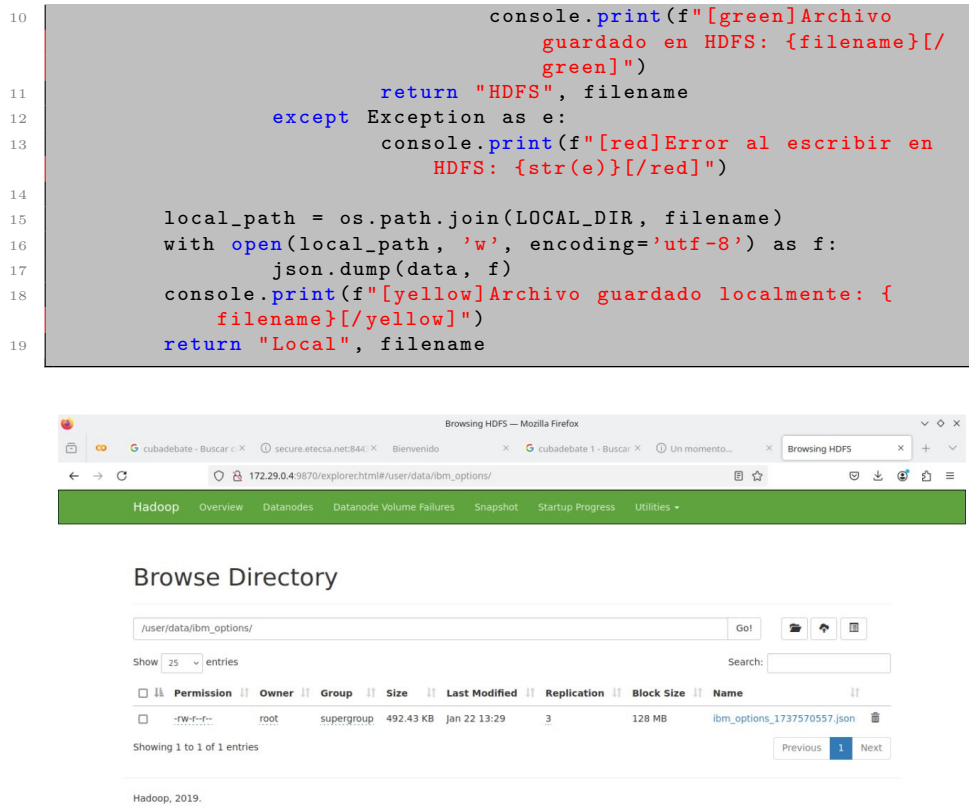


Figure 1: Esta imagen muestra toda la data almacenada en HDFS.

*IBM Options Data*

Timestamp	Number of Entries	Last File Saved
2025-01-23 10:32:13	1228	HDFS: ibm_options_1737646334.json
2025-01-23 10:33:16	1228	HDFS: ibm_options_1737646396.json
2025-01-23 10:34:19	1228	HDFS: ibm_options_1737646459.json
2025-01-23 10:35:21	1228	HDFS: ibm_options_1737646521.json

Figure 2: Esta imagen muestra en consola que la data se guardó correctamente.

## 4 Referencias

- [1]<https://kafka.apache.org/documentation/#gettingStarted>
- [2]<https://www.ibm.com/es-es/topics/hdfs>