

Fecha: 04/08/2025

Presentado por: Leonard Jose Cuenca Roa.

Ingeniería para el Procesado Masivo de Datos

[Actividad 2: Spark Streaming y Kafka](#)

Presentado por: Leonard Jose Cuenca Roa.

Fecha: 04/08/2025

Parte 1: Manejo de Spark Streaming

Como parte de la preparación de la clase, el profesor Abel nos proporcionó un paquete optimizado para el uso de HDFS y su entorno de software para el análisis de **Big Data**. Este ejercicio se ejecutará en un ambiente de Windows y, a continuación, mostraré el avance de la actividad con capturas de pantalla:

Paso 1: Se deja evidencia de la ejecución de los comandos Kafka para crear los topic para resolver la parte 1 de la actividad.

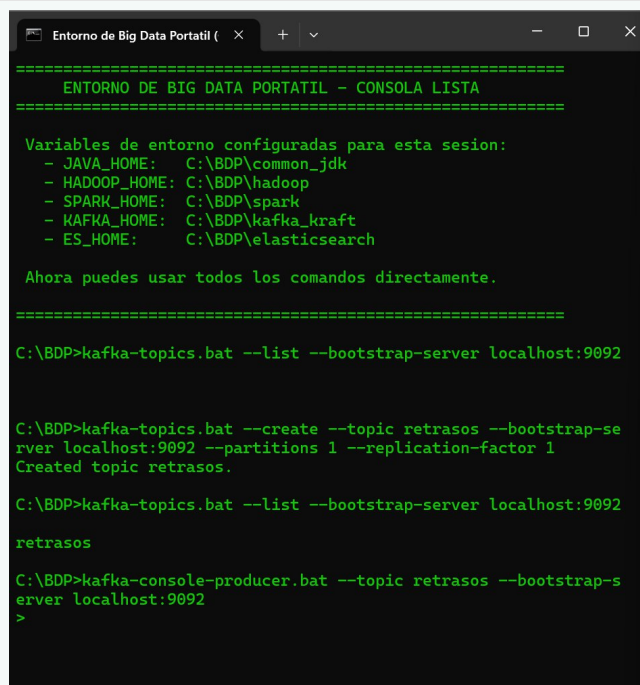
Descripción: Se ejecutaron los siguientes comandos para generar en el ambiente la comunicación Kafka y resolver los ejercicios de la parte 2, los comandos fueron los siguientes.

```
kafka-topics.bat --list --bootstrap-server localhost:9092
```

```
kafka-topics.bat --create --topic promedios --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

```
kafka-topics.bat --list --bootstrap-server localhost:9092
```

```
kafka-console-producer.bat --topic promedios --bootstrap-server localhost:9092
```



```
Entorno de Big Data Portatil ( x + v - □ x)
=====
ENTORNO DE BIG DATA PORTATIL - CONSOLA LISTA
=====

Variables de entorno configuradas para esta sesion:
- JAVA_HOME: C:\BDP\common_jdk
- HADOOP_HOME: C:\BDP\hadoop
- SPARK_HOME: C:\BDP\spark
- KAFKA_HOME: C:\BDP\kafka_kraft
- ES_HOME: C:\BDP\elasticsearch

Ahora puedes usar todos los comandos directamente.
=====

C:\BDP>kafka-topics.bat --list --bootstrap-server localhost:9092

C:\BDP>kafka-topics.bat --create --topic retrasos --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
Created topic retrasos.

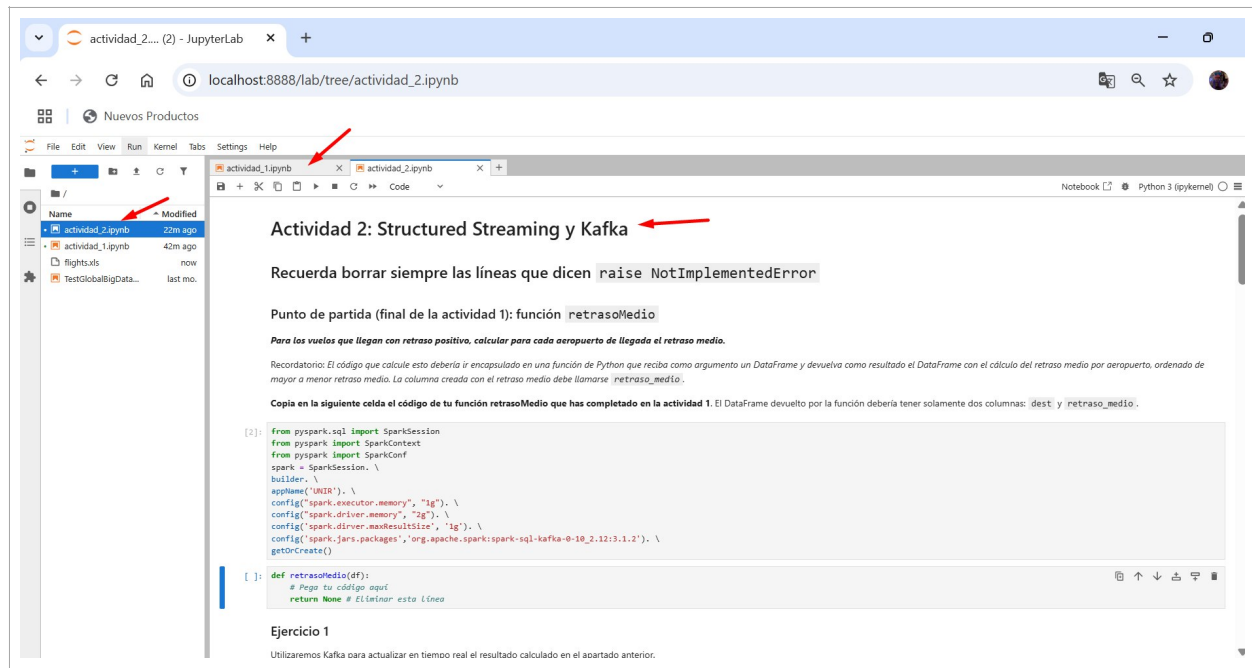
C:\BDP>kafka-topics.bat --list --bootstrap-server localhost:9092

retrasos

C:\BDP>kafka-console-producer.bat --topic retrasos --bootstrap-server localhost:9092
>
```

Paso 2: Ejercicio parte 1

Descripción: Como parte del seguimiento de la tarea de la parte 1, se sube el cuadernos al entorno para generar los script correspondiente y resolver los ejercicios



Parte 2: Manejo de Apache Kafka con *notebooks* de Jupyter.

Ejercicio 1

Descripción: Nos enfocamos en crear una conexión de **Spark Structured Streaming** con **Apache Kafka**. El objetivo principal es configurar un **Streaming DataFrame** para leer datos en tiempo real de un tópico de Kafka llamado retrasos.

Para lograr esto, se utilizó el método **readStream** de **Spark** y se especifica el formato "kafka". También se configuran las opciones de conexión, como la dirección de los brokers de Kafka y el tópico al que se debe suscribir, en este caso, **"retrasos"**.

El resultado de este ejercicio es un Streaming **DataFrame** (`retrasosStreamingDF`) con un esquema predefinido por **Spark** que incluye columnas como **key**, **value**, **topic**, **partition**, **offset**, **timestamp** y **timestampType**. De estas columnas, la más importante para el siguiente paso es **value**, ya que contiene los mensajes JSON en formato binario que serán procesados posteriormente.

Código:

```
# Reemplaza por el código correcto siguiendo las indicaciones anteriores
retrasosStreamingDF = None
```

```
## Metodo de comunicación
retrasosStreamingDF = (spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "127.0.0.1:9092")
    .option("subscribe", "retrasos")
    .load())
```

Evidencia:

Utilizaremos Kafka para actualizar en tiempo real el resultado calculado en el apartado anterior.

Para simplificar, asumimos que los mensajes leídos de Kafka tienen solamente dos campos que son los únicos necesarios para llevar a cabo la operación anterior: `dest` y `arr_delay`. La idea será crear un `Streaming Dataframe` para leer de Kafka, y después invocar a nuestra función `retratoMedio` pasándole como argumento. Vamos a leer del topic `retrasos` por lo que debes indicar esta opción a continuación.

Se pide crear, en la variable `retrasosStreamingDF`, un `Streaming Dataframe` leyendo de Apache Kafka, configurando las siguientes opciones:

- Usar la variable `readStream` (en lugar de `read` como solíamos hacer) interna de la `SparkSession` `spark`;
- Indicar que el formato es "kafka" con `.format("kafka")`;
- Indicar cuáles son los brokers de Kafka de los que vamos a leer y el puerto al que queremos conectarnos para leer (9092 es el que usa Kafka por defecto), con `.option("kafka.bootstrap.servers", "ciudadn_cluster-w-1:9092")`; De esa manera podremos leer el mensaje si el productor de Kafka lo envía a cualquiera de los dos brokers existentes que son los nodos del cluster identificados como `ciudadn_cluster-w-1` y `ciudadn_cluster-w-2`;
- Indicar que queremos suscribirnos al topic "retrasos" con `.option("subscribe", "retrasos")`;
- Finalmente ponemos `load()` para realizar la lectura.

```
[15]: # Amplias por el código correcto siguiendo las indicaciones anteriores
retrasosStreamingDF = None

# Retorno de comunicación
retrasosStreamingDF = (spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "127.0.0.1:9092")
    .option("subscribe", "retrasos")
    .load())

[16]: # Mostramos el esquema de este dataframe
types = retrasosStreamingDF.rdd.types
assert(retrasosStreamingDF.rdd.schema)
assert((types[0][0] == "key") & (types[0][1] == "binary"))
assert((types[1][0] == "value") & (types[1][1] == "binary"))
assert((types[2][0] == "topic") & (types[2][1] == "string"))
assert((types[3][0] == "partition") & (types[3][1] == "int"))
assert((types[4][0] == "offset") & (types[4][1] == "long"))
assert((types[5][0] == "timestamp") & (types[5][1] == "timestamp"))
```

Ejercicio 2: Retrasos

Descripción: Para iniciar con el ejercicio dos, se generan los esquemas para poder tratar la información usamos `StructType` para generar nuestro objeto, luego creamos nuestra variable `parsedDF` para complementar las columnas.

Evidencia:

```
parsedDF = retrasosStreamingDF \
    .withColumn("value", F.col("value").cast(StringType())) \
    .withColumn("parejas", F.from_json(F.col("value"), esquema)) \
    .withColumn("dest", F.col("parejas.dest")) \
    .withColumn("arr_delay", F.col("parejas.arr_delay"))

parsedDF.printSchema()

root
|-- key: binary (nullable = true)
|-- value: string (nullable = true)
|-- topic: string (nullable = true)
|-- partition: integer (nullable = true)
|-- offset: long (nullable = true)
|-- timestamp: timestamp (nullable = true)
|-- timestampType: integer (nullable = true)
|-- parejas: struct (nullable = true)
|   |-- dest: string (nullable = true)
|   |-- arr_delay: double (nullable = true)
|-- dest: string (nullable = true)
|-- arr_delay: double (nullable = true)
```

Se cargan los mensajes en usando la consola.

Evidencia:

The screenshot shows two terminal windows from the 'Entorno de Big Data Portatil' application. The left window displays environment variables and Kafka commands. The right window shows HDFS commands and their outputs.

```

Entorno de Big Data Portatil ( x + - □ x
=====
ENTORNO DE BIG DATA PORTATIL - CONSOLA LISTA
=====
Variables de entorno configuradas para esta sesion:
- JAVA_HOME: C:\BDP\common_jdk
- HADOOP_HOME: C:\BDP\hadoop
- SPARK_HOME: C:\BDP\spark
- KAFKA_HOME: C:\BDP\kafka_kraft
- ES_HOME: C:\BDP\elasticsearch

Ahora puedes usar todos los comandos directamente.
=====

C:\BDP>kafka-topics.bat --list --bootstrap-server localhost:9092

C:\BDP>kafka-topics.bat --create --topic retrasos --bootstrap-se
rver localhost:9092 --partitions 1 --replication-factor 1
Created topic retrasos.

C:\BDP>kafka-topics.bat --list --bootstrap-server localhost:9092
retrasos

C:\BDP>kafka-console-producer.bat --topic retrasos --bootstrap-s
erver localhost:9092
>{"dest": "GRX", "arr_delay": 2.6}
>{"dest": "MAD", "arr_delay": 5.4}
>{"dest": "GRX", "arr_delay": 1.5}
>{"dest": "MAD", "arr_delay": 20.0}

Entorno de Big Data Portatil ( x + - □ x
t
lsSnapshottableDir list all snapshottable dirs owned by the
current user Use -help to see

options
cacheadmin configure the HDFS cache
crypto configure HDFS encryption zones
mover run a utility to move block replicas across
ss storage types
storagepolicies list/get/set block storage policies

Most commands print help when invoked w/o parameters.

C:\BDP>dfs -put C:/flights.csv /CuencaRoaLeonardJose/w
"dfs" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\BDP>dfs -put C:/flights.csv /CuencaRoaLeonardJose/wo
"dfs" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\BDP>dfs -put C:/flights.csv /CuencaRoaLeonardJose/wo

C:\BDP>hdfs dfs -put C:/flights.csv /CuencaRoaLeonardJose/
put: '/CuencaRoaLeonardJose/flights.csv': File exists

C:\BDP>hdfs dfs -ls /CuencaRoaLeonardJose
Found 1 items
-rw-r--r-- 1 Kennta supergroup 11244080 2025-08-03 17:56 /C
uencaRoaLeonardJose/flights.csv

C:\BDP>

```

Los scripts aborda la tarea de procesar datos en tiempo real mediante la integración de dos tecnologías clave en el ámbito de Big Data: Apache Spark Structured Streaming y Apache Kafka. El proceso inicia con la creación de un Streaming DataFrame (`retrasosStreamingDF`), el cual se configura para establecer una conexión de escucha continua con el tópico de Kafka llamado `retrasos`. Esta conexión es la base para recibir los mensajes en tiempo real, que se presentan en formato JSON dentro de una columna llamada `value`.

Posteriormente, el script se enfoca en la estructuración de los datos recibidos. Para ello, los mensajes en formato binario de la columna `value` son convertidos a tipo `string`. Una vez convertidos, se utiliza la función `from_json` para analizar el contenido del JSON, extrayendo de manera ordenada los campos `dest` (aeropuerto de destino) y `arr_delay` (retraso en la llegada). Esta operación transforma los datos brutos en un formato estructurado y legible, que es fundamental para los cálculos posteriores.

Finalmente, el script aplica una lógica de negocio para procesar los datos estructurados. Utiliza una función (`retrasoMedio`) para calcular el retraso medio de los vuelos por aeropuerto, considerando solo aquellos que tienen un retraso positivo. Los resultados de esta agregación se almacenan en una tabla en memoria (`retrasosAgg`) que se actualiza dinámicamente con cada nuevo mensaje recibido. Esto permite la visualización en tiempo real de los resultados, demostrando cómo Spark Structured Streaming puede procesar y analizar flujos de datos a medida que llegan.