

Unidad 2 - Tarea 3 - Procesamiento de Datos con Apache Spark

Nombre del estudiante

Giovanny Alejandro Pardo

Grupo:

Big Data (202016911_27)

Tutora

Sandra Milena Patino Avella

Universidad Nacional Abierta y a Distancia-UNAD

Escuela de ciencias básicas, tecnológicas e ingeniería

Ingeniería de sistemas

Palmira – octubre 17 del 2024

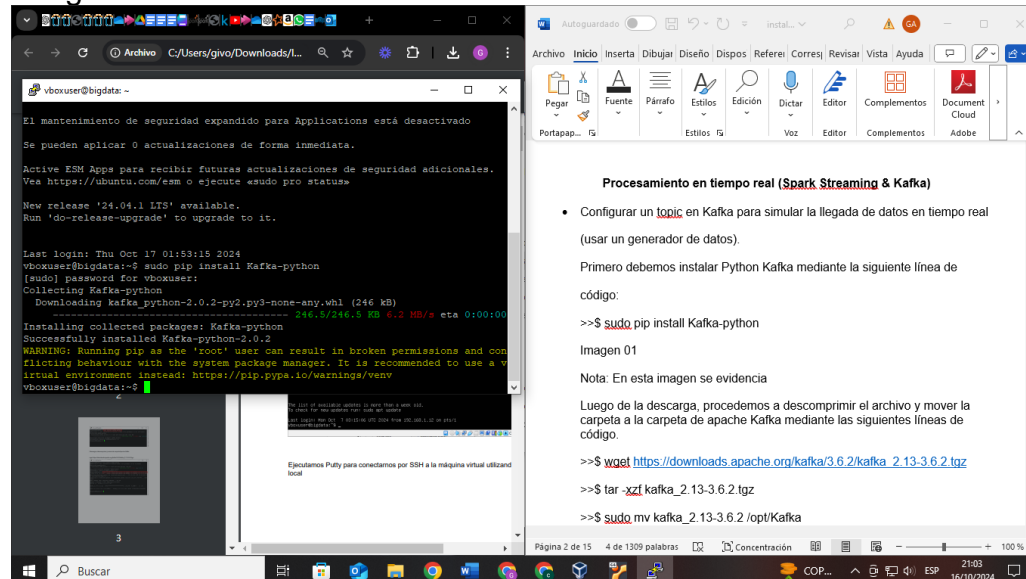
Procesamiento en tiempo real (Spark Streaming & Kafka)

- Configurar un topic en Kafka para simular la llegada de datos en tiempo real (usar un generador de datos).

Primero debemos instalar Python Kafka mediante la siguiente línea de código:

```
>>$ sudo pip install Kafka-python
```

Imagen 01

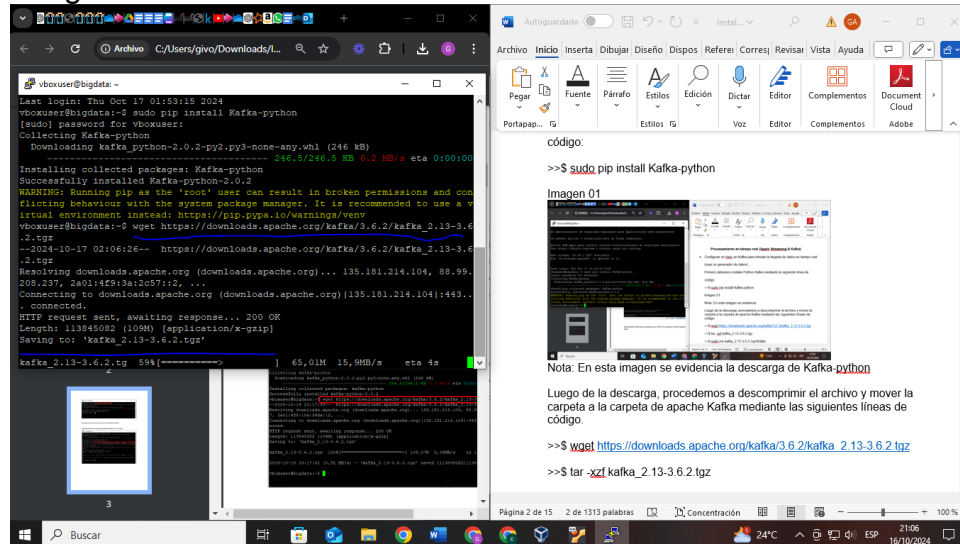


Nota: En esta imagen se evidencia la descarga de Kafka-python

Luego de la descarga, procedemos a descomprimir el archivo y mover la carpeta a la carpeta de apache Kafka mediante las siguientes líneas de código.

```
>>$ wget https://downloads.apache.org/kafka/3.6.2/kafka\_2.13-3.6.2.tgz
```

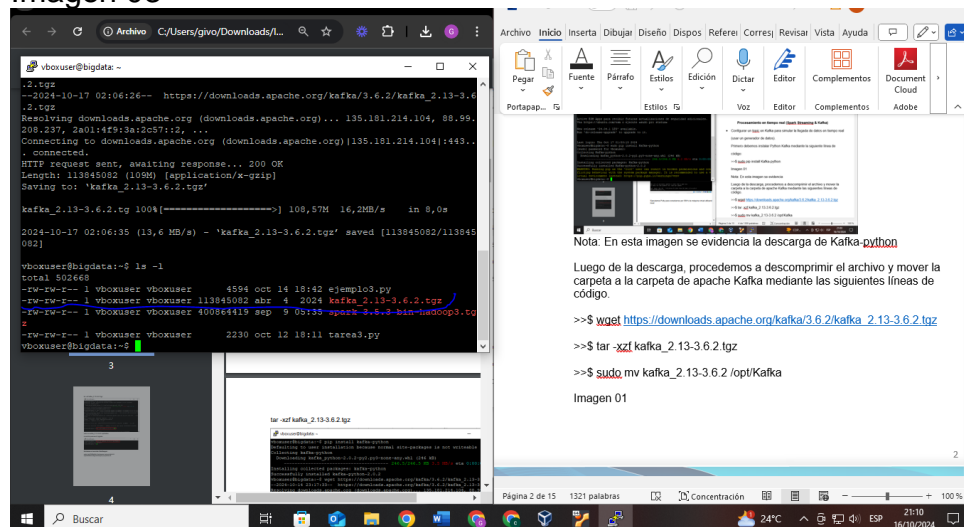
Imagen 02



Nota: En esta imagen se evidencia que se inicia la descarga del archivo comprimido.

Con el comando `ls -l` podemos ver el archivo descargado en .tgz

Imagen 03

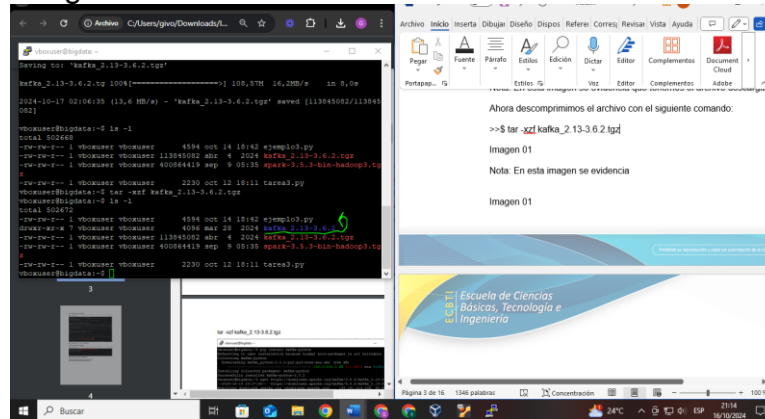


Nota: En esta imagen se evidencia que tenemos el archivo descargado.

Ahora descomprimos el archivo con el siguiente comando:

```
>>$ tar -xzf kafka_2.13-3.6.2.tgz
```

Imagen 04

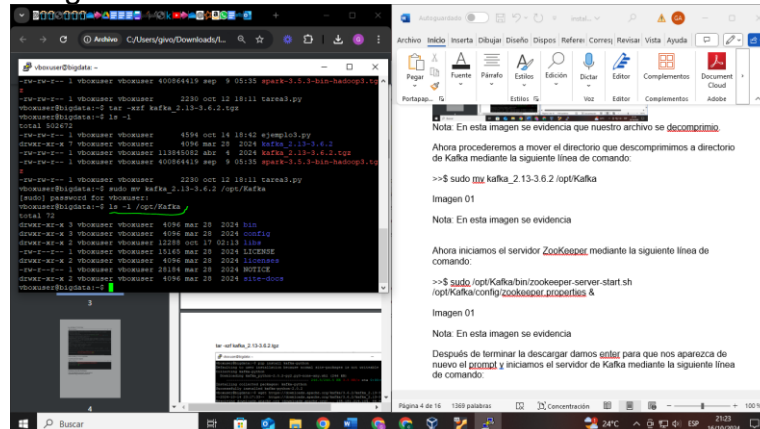


Nota: En esta imagen se evidencia que nuestro archivo se descomprimio.

Ahora procederemos a mover el directorio que descomprimimos a directorio de Kafka mediante la siguiente línea de comando:

```
>>$ sudo mv kafka_2.13-3.6.2 /opt/Kafka
```

Imagen 05

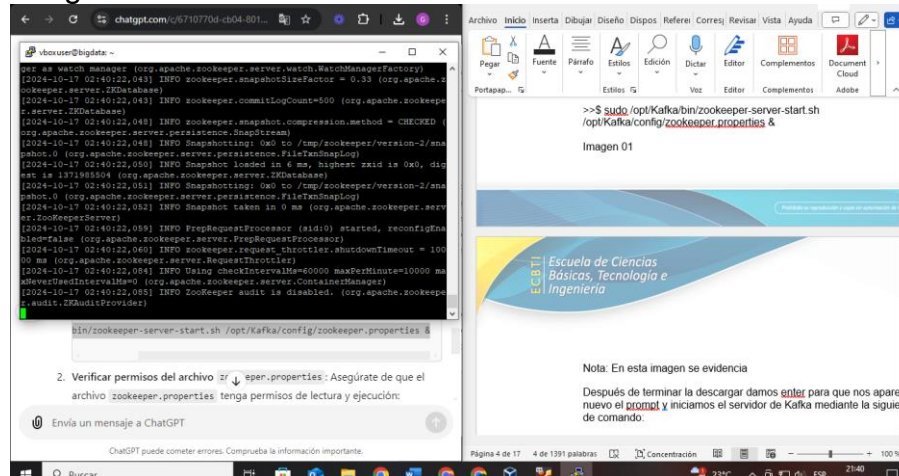


Nota: En esta imagen se evidencia los archivos que existen en el directorio /opt/Kafka hay 72 registros, pero solo tenemos como usuario permisos para cer alrededor de 7.

Ahora iniciamos el servidor ZooKeeper mediante la siguiente línea de comando:

```
>>$ /opt/Kafka/bin/zookeeper-server-start.sh /opt/Kafka/config/zookeeper.properties &
```

Imagen 06

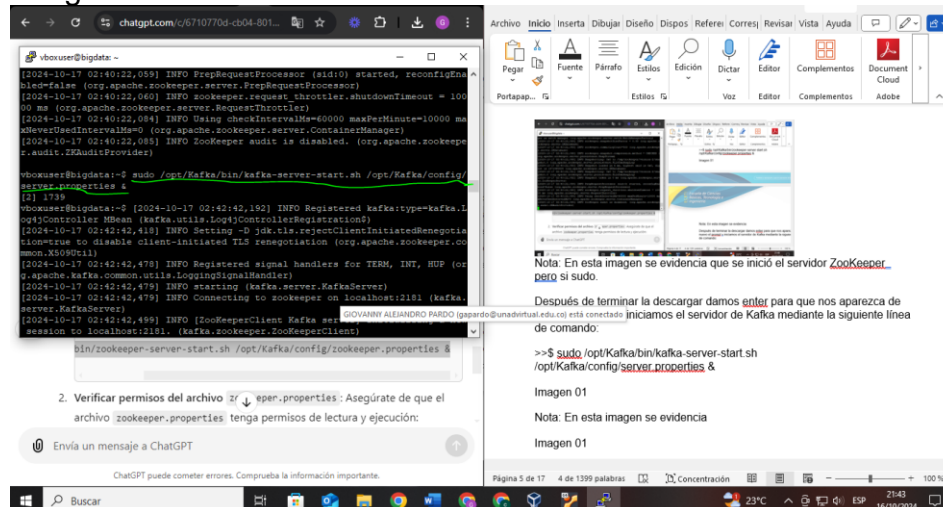


Nota: En esta imagen se evidencia que se inició el servidor ZooKeeper pero si sudo.

Después de terminar la descargar damos enter para que nos aparezca de nuevo el prompt e iniciamos el servidor de Kafka mediante la siguiente línea de comando:

```
>>$ sudo /opt/Kafka/bin/kafka-server-start.sh
/opt/Kafka/config/server.properties &
```

Imagen 07



Nota: En esta imagen se evidencia que se inicia el servidor Kafka.

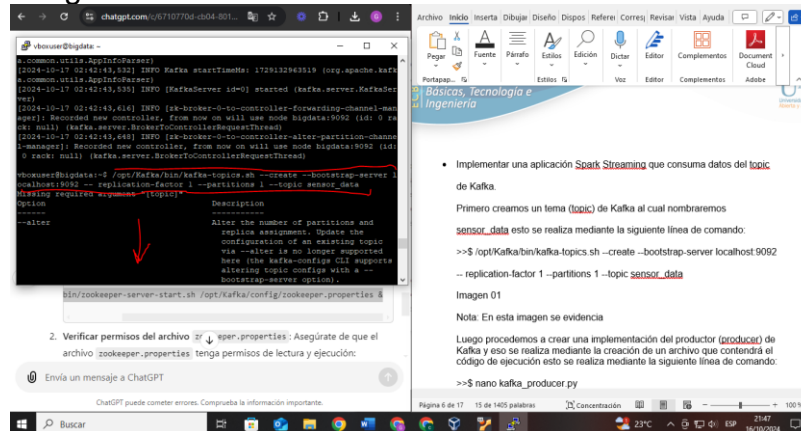
- Implementar una aplicación Spark Streaming que consuma datos del topic de Kafka.

Primero creamos un tema (topic) de Kafka al cual nombraremos

sensor_data esto se realiza mediante la siguiente línea de comando:

```
>>$ /opt/Kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092
-- replication-factor 1 --partitions 1 --topic sensor_data
```

Imagen 08



Nota: En esta imagen se evidencia que se inicia la creación del tema (topic).

Luego procedemos a crear una implementación del productor (producer) de Kafka y eso se realiza mediante la creación de un archivo que contendrá el código de ejecución esto se realiza mediante la siguiente línea de comando:

```
>>$ nano kafka_producer.py
```

Luego copiamos estas líneas de comandos:

```
import time
import json
import random
from kafka import KafkaProducer
def generate_sensor_data():
    return {
        "sensor_id": random.randint(1, 10),
```

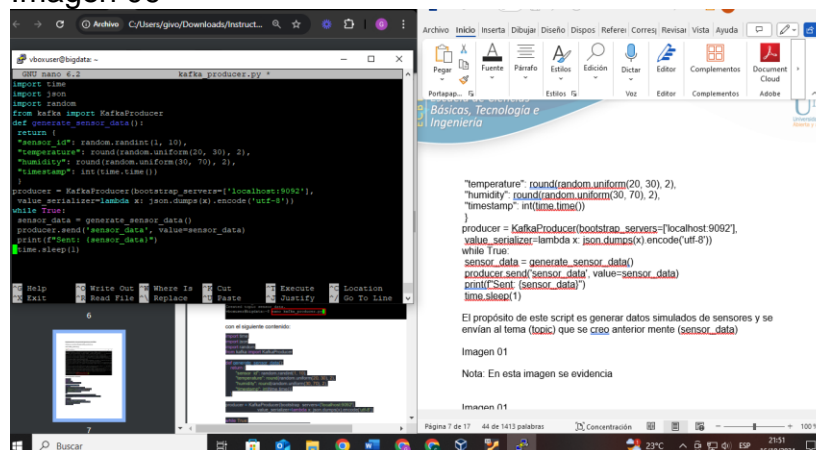
```

"temperature": round(random.uniform(20, 30), 2),
"humidity": round(random.uniform(30, 70), 2),
"timestamp": int(time.time())
}
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
value_serializer=lambda x: json.dumps(x).encode('utf-8'))
while True:
    sensor_data = generate_sensor_data()
    producer.send('sensor_data', value=sensor_data)
    print(f"Sent: {sensor_data}")
    time.sleep(1)

```

El propósito de este script es generar datos simulados de sensores y se envían al tema (topic) que se creó anterior mente (sensor_data)

Imagen 09



Nota: En esta imagen se evidencia que creamos el script de nombre “Kafka_producer.py”.

- Realizar algún tipo de procesamiento o análisis sobre los datos en tiempo real (contar eventos, calcular estadísticas, etc.).

Ahora procederemos a crear un consumidor (consumer) utilizando Spark Streaming para procesar los datos en tiempo real, para ello crearemos un

archivo llamado `spark_streaming_consumer.py` mediante la siguiente línea de comando:

```
>>$ nano spark_streaming_consumer.py
```

y le agregamos las siguientes líneas de instrucciones:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, window
from pyspark.sql.types import StructType, StructField, IntegerType,
FloatType, TimestampType
import logging
# Configura el nivel de log a WARN para reducir los mensajes INFO
spark = SparkSession.builder \
    .appName("KafkaSparkStreaming") \
    .getOrCreate()
spark.sparkContext.setLogLevel("WARN")
# Definir el esquema de los datos de entrada
schema = StructType([
    StructField("sensor_id", IntegerType()),
    StructField("temperature", FloatType()),
    StructField("humidity", FloatType()),
    StructField("timestamp", TimestampType())
])
# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("SensorDataAnalysis") \
    .getOrCreate()
# Configurar el lector de streaming para leer desde Kafka
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "sensor_data") \
    .load()
# Parsear los datos JSON
parsed_df = df.select(from_json(col("value").cast("string"),
schema).alias("data")).select("data.*")
# Calcular estadísticas por ventana de tiempo
windowed_stats = parsed_df \
    .groupBy(window(col("timestamp"), "1 minute"), "sensor_id") \
    .agg({"temperature": "avg", "humidity": "avg"})
```

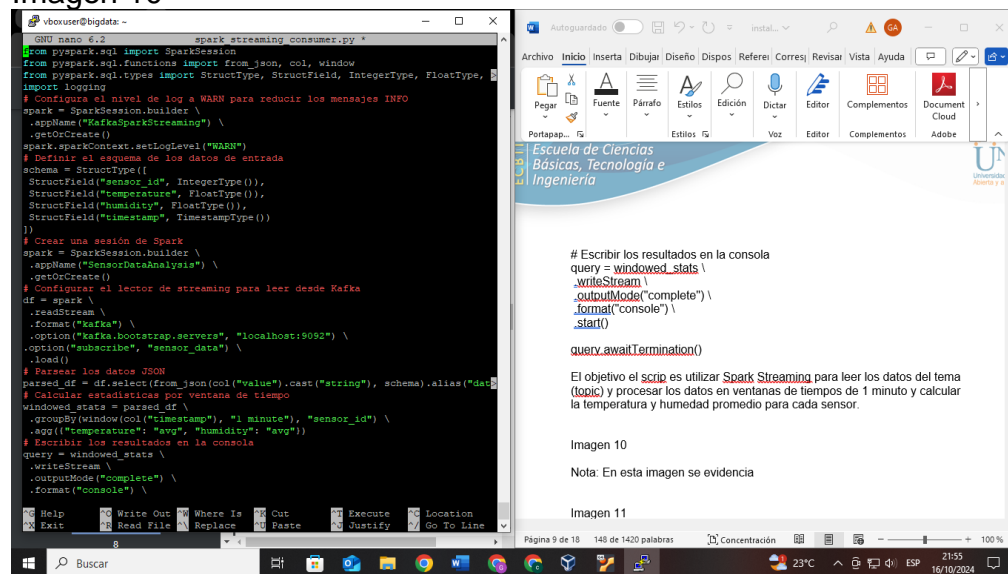

Escribir los resultados en la consola

```
query = windowed_stats \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()
```

```
query.awaitTermination()
```

El objetivo del script es utilizar Spark Streaming para leer los datos del tema (topic) y procesar los datos en ventanas de tiempos de 1 minuto y calcular la temperatura y humedad promedio para cada sensor.

Imagen 10

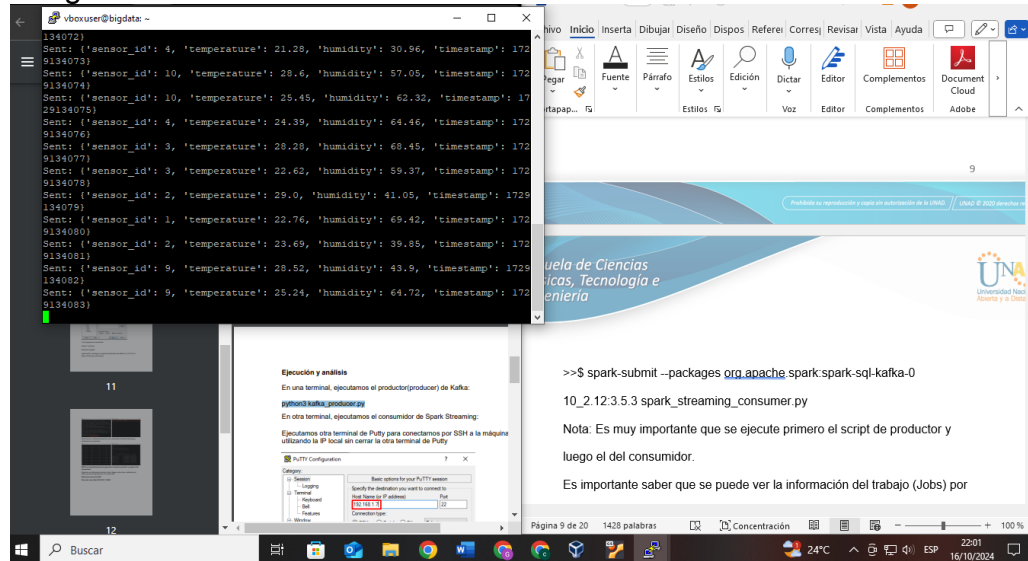


Nota: En esta imagen se evidencia la creación del script de nombre “spark_streaming_consumer.py”

- Visualizar los resultados del procesamiento en tiempo real.

Ahora debemos abrir un termina mediante putty y ejecutamos el script de nombre “python3 kafka_producer.py”

Imagen 12



Nota: En esta imagen se evidencia que se inicia el script en ejecución infinita.

Mientras que en un nuevo terminal se ejecutara el consumidor de Spark

Streaming: mediante la siguiente línea de comando:

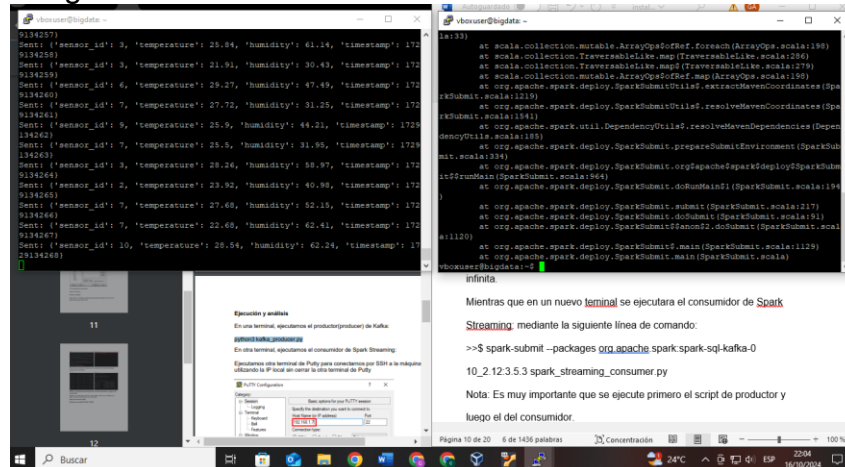
```
>>$ spark-submit --packages org.apache.spark:spark-sql-kafka-0
```

```
10_2.12.3.5.3 spark_streaming_consumer.py
```

Nota: Es muy importante que se ejecute primero el script de productor y

luego el del consumidor.

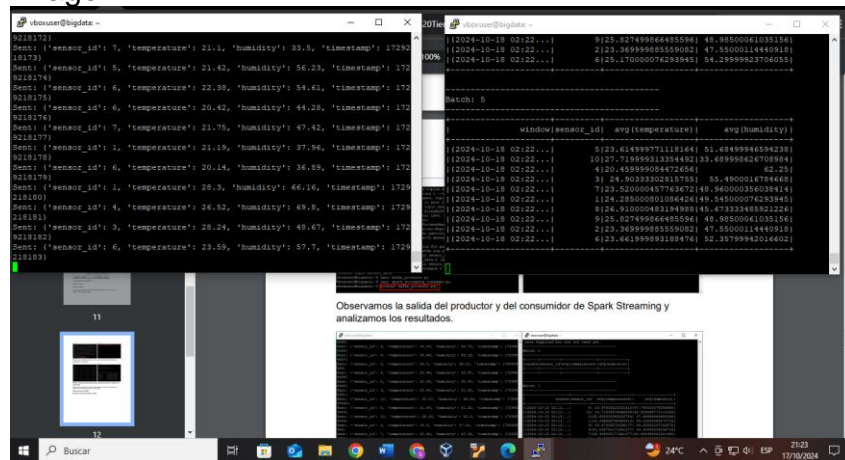
Imagen 13



Nota: En esta imagen se evidencia que a la izquierda se ejecutó primero el script productor y a la derecha se ejecutó después el script consumidor.

Es importante saber que se puede ver la información del trabajo (Jobs) por el puerto 4040 y usando la ip del equipo Ubuntu Server.

Imagen 14



Nota: En esta imagen se evidencia que tenemos los datos que se estan procesando la lectura de los sensores en la terminal derecha.

Ahora en el interfaz de control de spark donde tenemos nuestro job ejecutándose vamos a ver las diferentes pestañas que nos mostraran información relevante del proceso de análisis de los datos en tiempo real.

En esta pestaña se puede apreciar las consultas SQL y DataFrame que se esta ejecutando se puede apreciar la lista de consultas que se an enviado o completado mostrando detalles sobre el tiempo de ejecución, duración y el estado del trabajo.

The screenshot displays the 'Completed Queries (11)' section of the SQL DataFrame application. It features a table with 11 rows, each representing a query. The columns are: ID, Description, Submitted, Duration, Running Job ID(s), Succeeded Job ID(s), Failed Job ID(s), and Job Execution ID(s). The queries are numbered 10 through 20. The 'Submitted' column shows dates from 2024-10-18 02:24:00 to 2024-10-18 02:39:00. The 'Duration' column shows values like 2s, 3s, 4s, and 5s. The 'Running Job ID(s)' column contains values like 'dbench'. The 'Succeeded Job ID(s)' column contains values like 'dbench'. The 'Failed Job ID(s)' column is empty. The 'Job Execution ID(s)' column contains values like 'dbench'. The interface includes a search bar at the top, a navigation bar with tabs for 'SQL', 'DataFrame', 'Database', 'Tools', and 'Help', and a footer with a Windows taskbar showing the date and time as 21:04 on 10/18/2024.

ID	Description	Submitted	Duration	Running Job ID(s)	Succeeded Job ID(s)	Failed Job ID(s)	Job Execution ID(s)
10	SQL DataFrame - Running Query 10	2024-10-18 02:24:00	2s	dbench			dbench
11	SQL DataFrame - Running Query 11	2024-10-18 02:24:00	2s	dbench			dbench
12	SQL DataFrame - Running Query 12	2024-10-18 02:24:00	2s	dbench			dbench
13	SQL DataFrame - Running Query 13	2024-10-18 02:24:00	2s	dbench			dbench
14	SQL DataFrame - Running Query 14	2024-10-18 02:24:00	2s	dbench			dbench
15	SQL DataFrame - Running Query 15	2024-10-18 02:24:00	2s	dbench			dbench
16	SQL DataFrame - Running Query 16	2024-10-18 02:24:00	2s	dbench			dbench
17	SQL DataFrame - Running Query 17	2024-10-18 02:24:00	2s	dbench			dbench
18	SQL DataFrame - Running Query 18	2024-10-18 02:24:00	2s	dbench			dbench
19	SQL DataFrame - Running Query 19	2024-10-18 02:24:00	2s	dbench			dbench
20	SQL DataFrame - Running Query 20	2024-10-18 02:39:00	5s	dbench			dbench

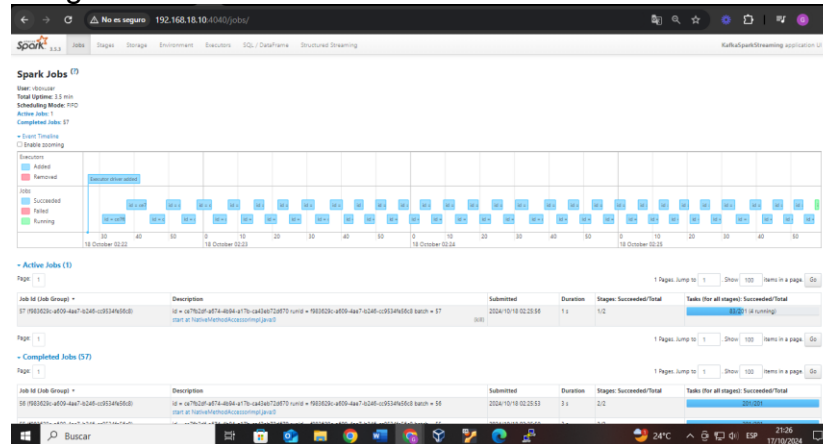
Procedemos a abrir el queries 1.

Nota: En esta imagen se evidencia todo el proceso de análisis de datos.

Jobs:

Aquí podemos ver una lista de todos los trabajos de Spark que se han ejecutado, se están ejecutando o los que han fallado también se pueden apreciar detalles como son el estado del trabajo, la cantidad de tareas completadas y el progreso general del job.

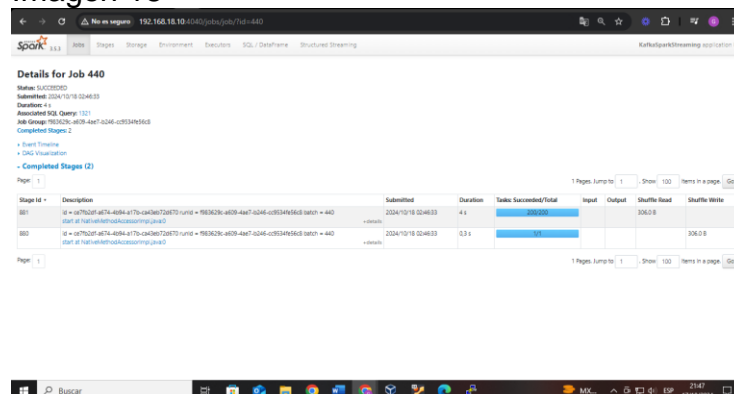
Imagen 17



Nota: En esta imagen se evidencia los trabajos ejecutados en el Job.

Abrimos el Job 440 para ver sus detalles.

Imagen 18



Nota: En esta imagen se evidencia con más claridad los detalles del job 440.

Stages:

En esta pestaña se puede apreciar las diferentes etapas de un job (trabajo) de spark, el cual se divide en tareas. Cada trabajo en Spark se compone de etapas las cuales permiten monitorear los progresos y los detalles de cada etapa, así como tareas pendientes y fallidas.

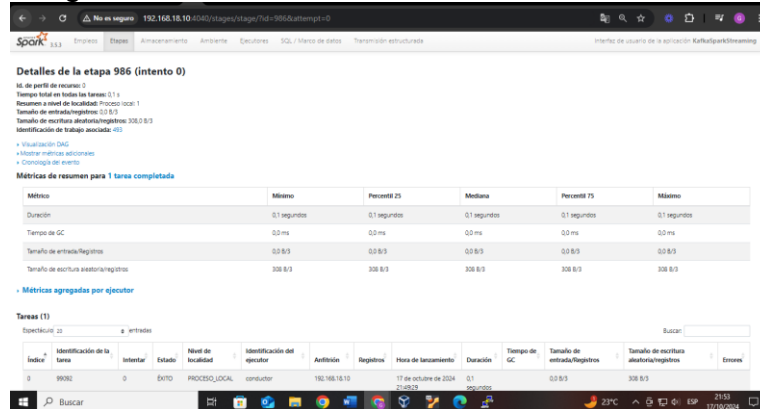
Imagen 19

Stage ID	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
147	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 73 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:47	3 s	186/200 (93%)			311.0 B	
148	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 73 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:47	0.1 s	1/1			311.0 B	
149	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 72 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:44	3 s	200/200			312.0 B	
148	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 72 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:44	0.2 s	1/1			312.0 B	
143	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 71 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:41	3 s	200/200			412.0 B	
142	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 71 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:41	0.5 s	1/1			412.0 B	
141	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 70 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:37	3 s	200/200			208.0 B	
140	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 70 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:37	0.5 s	1/1			208.0 B	
139	id = ca7b2d4f-a674-4b84-a17b-ca43e72d670 runid = 993629c-a509-4ae7-b246-cc95349e56d batch = 69 start at NativeMethodAccessImpl.java0	2024/10/18 02:28:33	4 s	200/200			208.0 B	

Nota: En esta imagen se evidencia los stages o etapas del job.

Si abrimos la etapa 986 podemos observar varios datos como el tiempo de ejecución fue de 0.1 segundo, las métricas de resumen entre otros datos importante.

Imagen 20



Detalles de la etapa 986 (intento 0)

Id. de perfil de recurso: 0
 Tiempo total en todos los tareas: 0.1 s
 Resumen a nivel de localidad: Tiempo total: 1
 Tamaño de entrada/registros: 0.0 B
 Tamaño de escritura a destino/registros: 0.0 B
 Identificación de trabajo asociada: 453

+ Visualización DAG
 + Mostrar métricas adicionales
 + Cronología del evento

Métricas de resumen para 1 tarea completada

Métrica	Mínimo	Percentil 25	Mediana	Percentil 75	Máximo
Duración	0.1 segundos	0.1 segundos	0.1 segundos	0.1 segundos	0.1 segundos
Tiempo de GC	0.0 ms	0.0 ms	0.0 ms	0.0 ms	0.0 ms
Tamaño de entrada/registros	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Tamaño de escritura a destino/registros	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

+ Métricas agregadas por ejecutor

Tareas (1)

Espectador: 23 Entidad:

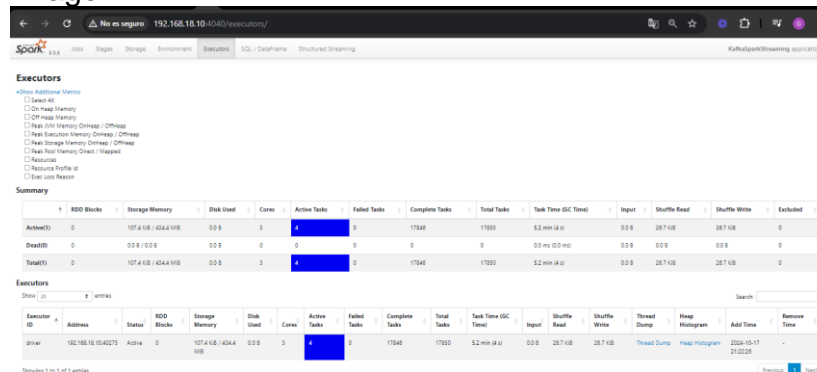
Id	Identificación de la tarea	Intentar	Estado	Nivel de localidad	Identificación del ejecutor	Asignación	Registros	Hora de lanzamiento	Duración	Tiempo de GC	Tamaño de entrada/registros	Tamaño de escritura a destino/registros	Errores
9860	0	EXITO	PROCESO LOCAL	completer	192.168.16.10	17 de febrero de 2024 14:05:28	0	0.1	0.000000	0.0 B	0.0 B	0.0 B	

Nota: En esta imagen se evidencia la etapa 986 y sus detalles.

Executor:

En esta pestaña nos proporciona detalles sobre cada uno de los ejecutores (executors) que se están procesando la tarea de Spark, nos muestra estadísticas tales como el uso de memoria, la cantidad de tareas que ha procesado cada ejecutor y el tiempo de actividad.

Imagen 21



Executors

Ver: Resumen de recursos

- ☐ Show All
- ☐ On-Heap Memory
- ☐ Off-Heap Memory
- ☐ Peak On-Heap Memory (Off-Heap)
- ☐ Peak Execution Memory (Off-Heap)
- ☐ Peak Storage Memory (Off-Heap)
- ☐ Peak Root Memory (Off-Heap)
- ☐ Resources
- ☐ Resource Profile
- ☐ Error Logs Reason

Summary

	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	0	17345	17350	0.0 min (4.0)	0.0	26.7 GB	26.7 GB	0
Dead(0)	0	0	0	0	0.0 ms (0.0 ms)	0.0	0.0	0.0	0
Total(1)	0	0	17345	17350	0.0 min (4.0)	0.0	26.7 GB	26.7 GB	0

Executors

Executor ID	Address	Status	Heap Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram	Add Time	Remove Time
driver	192.168.16.10:40275	Active	107.4 GB / 454.4 GB	0.0 B	3	1	0	17345	17350	0.0 min (4.0)	0.0	26.7 GB	26.7 GB	Thread Dump	Heap Histogram	2024-02-17 14:05:28	-

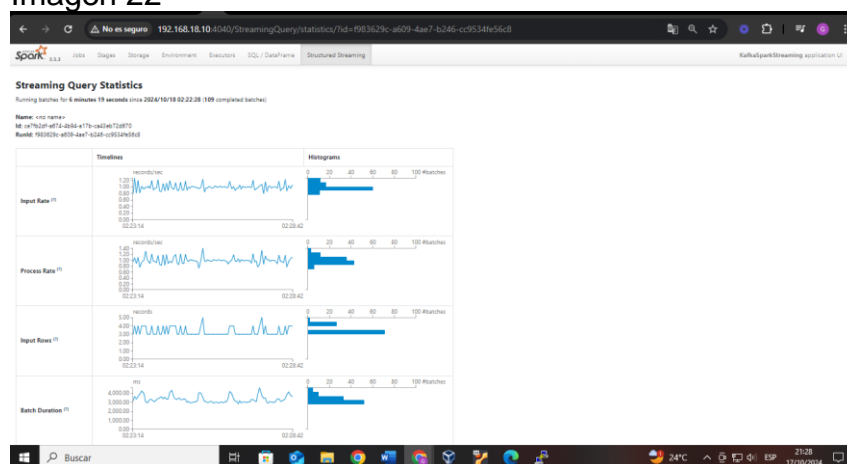
Showing 1 to 1 of 1 entries

Nota: En esta imagen se evidencia detalles de los ejecutores.

Structured Streaming:

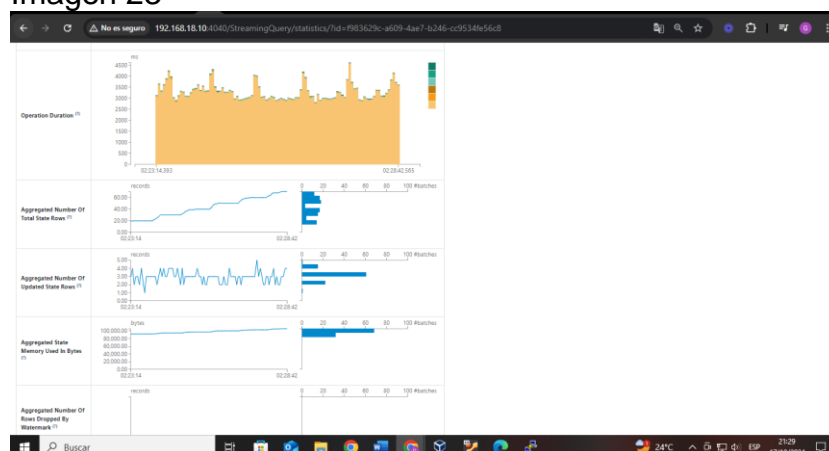
Esta pestaña nos muestra informes sobre las consultas de streaming en ejecución, lactancia, tasa de entrada, duración del lote, duración de la operación, el rendimiento y otros detalles de relevancia.

Imagen 22



Nota: En esta imagen se evidencia las estadísticas de consulta de streaming.

Imagen 23



Nota: En esta imagen se evidencia las estadísticas de consulta de streaming.