# Comprehensive Data Pipeline Solutions Using Spark, Kafka, and AWS

Ronak Sengupta

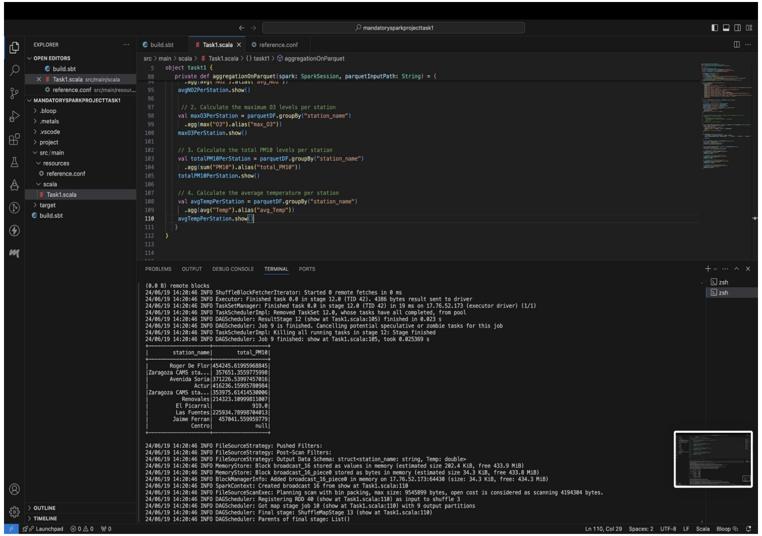
Senior Software Engineer 2

Payoda Technologies

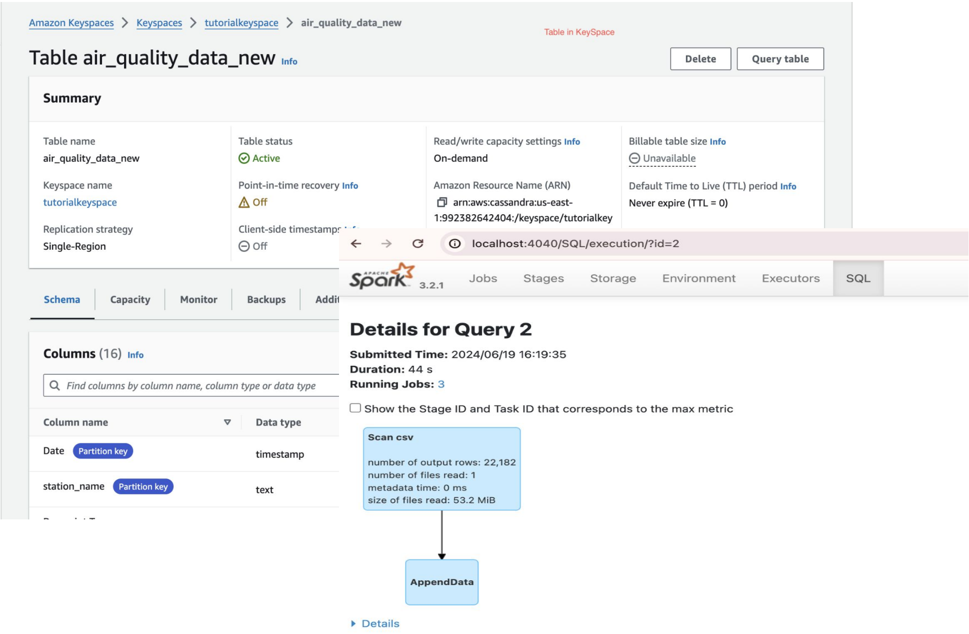
# Scenario 1 - Data Pipeline for performing Data conversions and performing aggregation operations

**Description:** In this project, I designed a data pipeline to analyze air quality data by seamlessly reading from AWS S3, writing to AWS Keyspaces for storage, converting and storing as Parquet format in S3, and performing crucial aggregation operations, designed by Apache Spark for efficient data processing. [**Code Link:-** https://github.com/r1999-ron/SparkScenario1Task]

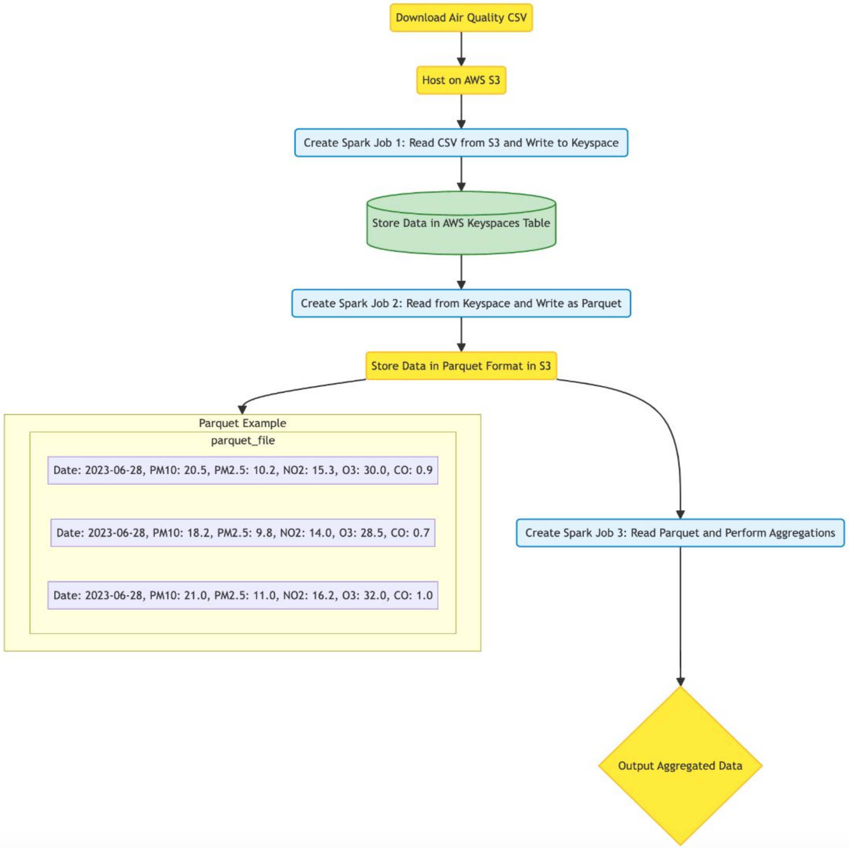
## Screenshot of aggregate operations



## keyspace table screenshot & DAG Output



# Flow Diagram - Scenario 1:

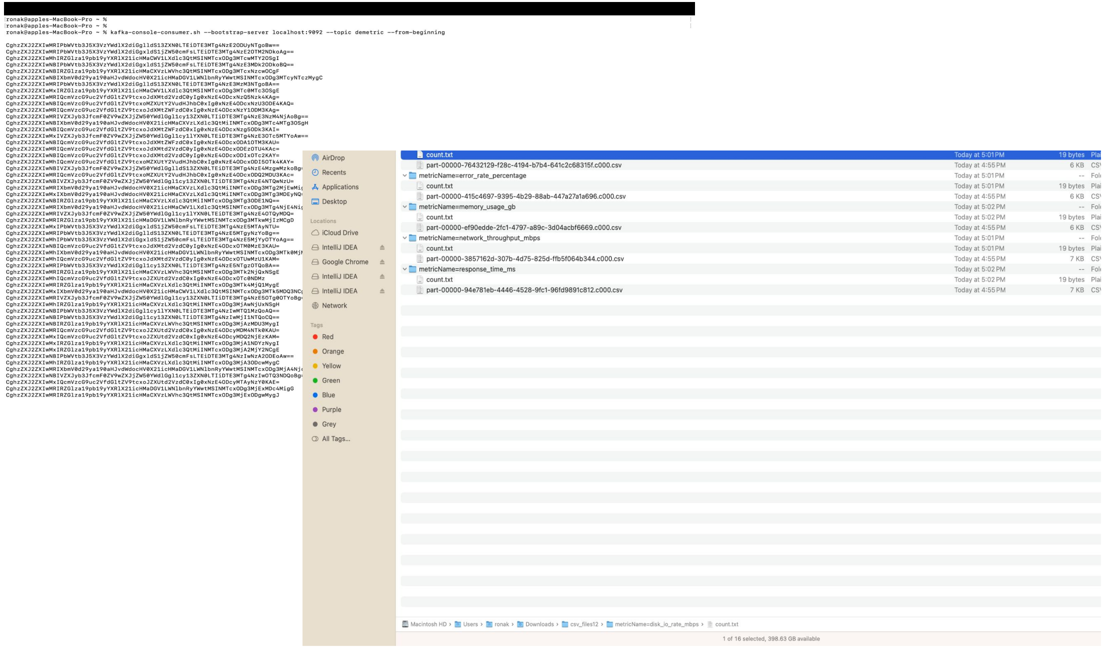


# Scenario 2 - Streaming JOB using Kafka , Spark and Protobuf

**Description:** In Scenario 2, a streaming pipeline using AKKA, Apache Spark, Protobuf, Spark Streaming, and Apache Kafka was implemented. An AKKA microservice generates JSON server metrics data posted to Kafka every 5 seconds. Spark Streaming converts JSON to Protobuf and publishes to another Kafka topic, then deserializes Protobuf to CSV files based on metric type. Spark Jobs aggregate these CSV files for comprehensive analysis, leveraging Spark and Kafka for scalable, fault-tolerant data processing. [**Code Link:-** https://github.com/r1999-ron/SparkScenario2Task & https://github.com/r1999-ron/SparkScenario2.2Task]

## Screenshot of akka kafka topic

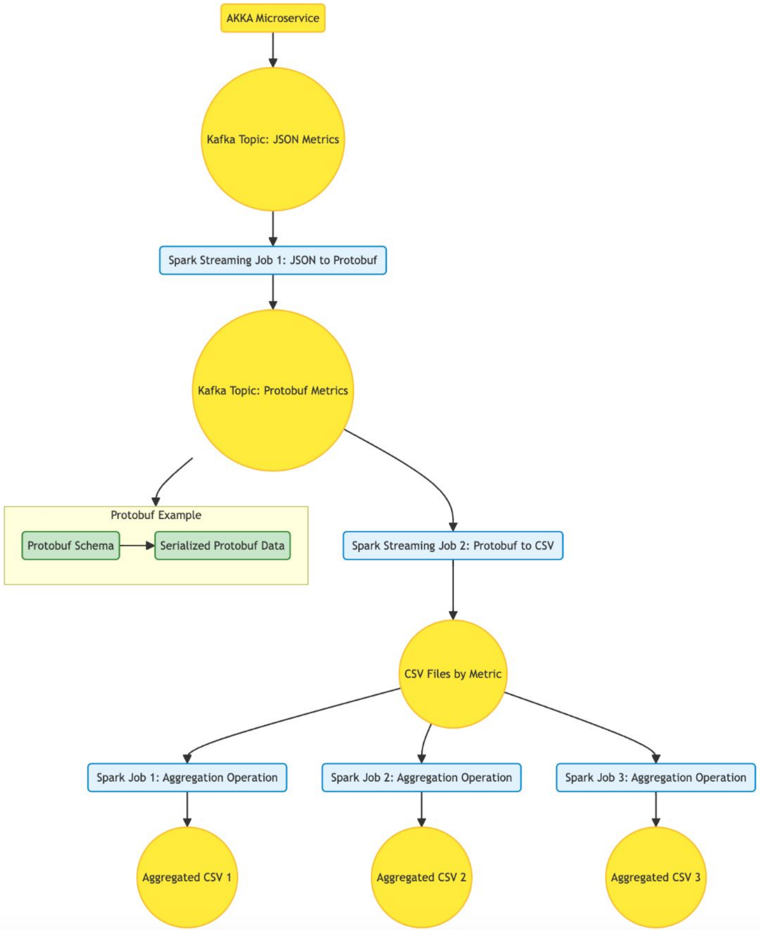

## Screenshot of binary data(protobuf format) & output csv screenshots




## Aggregate Output

| 1 | server_name | total_sum | count | average |
|---|---|---|---|---|
| 2 | server02 | 436 | 8 | 54.5 |

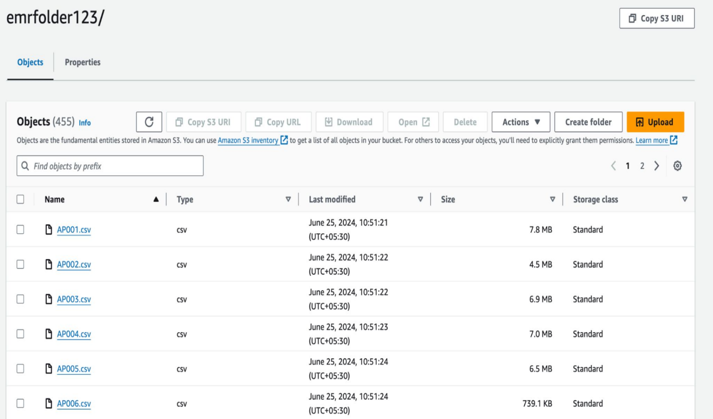
# Flow Diagram - Scenario 2:



# Scenario 3 - Executing Jobs on EMR

**Description: -** Scenario 3 utilizes AWS EMR, RDBMS, S3 Bucket, Spark, and Scala for data processing. It begins with setting up an EMR Cluster and Studio with a notebook. The dataset from Kaggle is stored in an S3 bucket, processed using an EMR Notebook to convert files to Parquet format and store them in another S3 bucket. Ten Spark Jobs on EMR perform operations like filtering and aggregation on the Parquet data, with results stored in MySQL tables for structured analysis and management. [**Notebook Link:-** https://github.com/r1999-ron/SparkMandatoryTask3/blob/main/SparkMandatory3Task.ipynb]

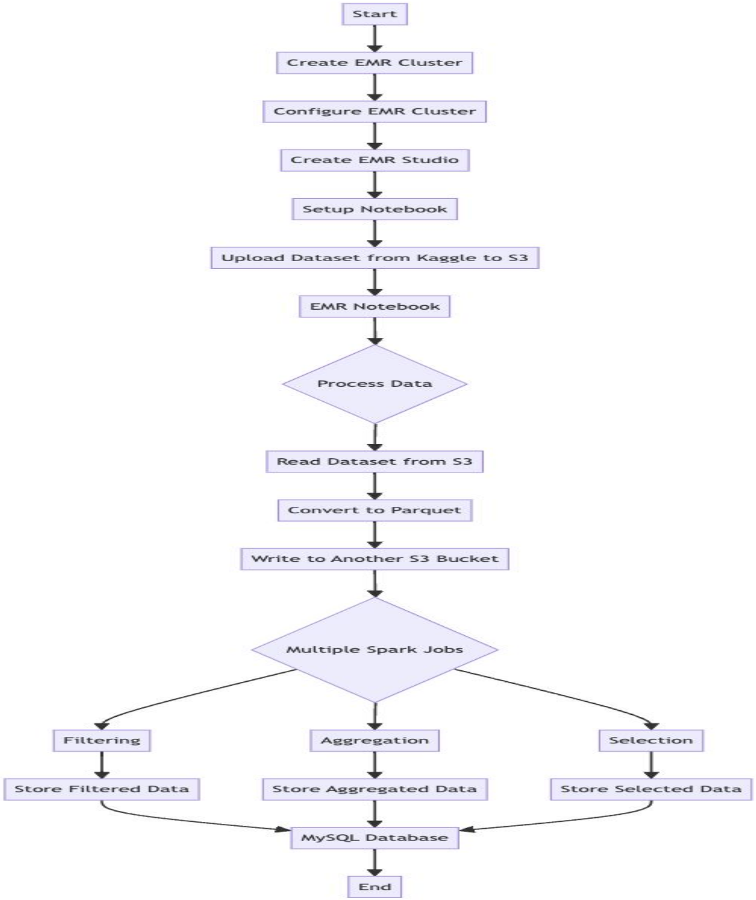
## Screenshot of S3 bucket:




## Spark Job filtration screenshot

```
mysql> select * from avg_ws_df limit 0,10;
+---------------------+---------------------+
| From Date           | Avg_WS              |
+---------------------+---------------------+
| 2023-02-10 10:00:00 |   266.8008053691275 |
| 2023-02-10 23:00:00 | 223.77164285714284 |
| 2023-02-13 16:00:00 |   244.744060402684б |
| 2023-02-17 10:00:00 |    249.421821192053 |
| 2023-03-03 05:00:00 | 221.19941379310345 |
| 2023-03-06 13:00:00 |   271.3822580645161 |
| 2023-02-03 01:00:00 | 220.89655913978498 |
| 2023-02-04 11:00:00 |   277.2291467576792 |
| 2023-01-14 17:00:00 | 220.35160142348755 |
| 2023-01-29 10:00:00 |   235.0301742160279 |
+---------------------+---------------------+
10 rows in set (0.29 sec)
```

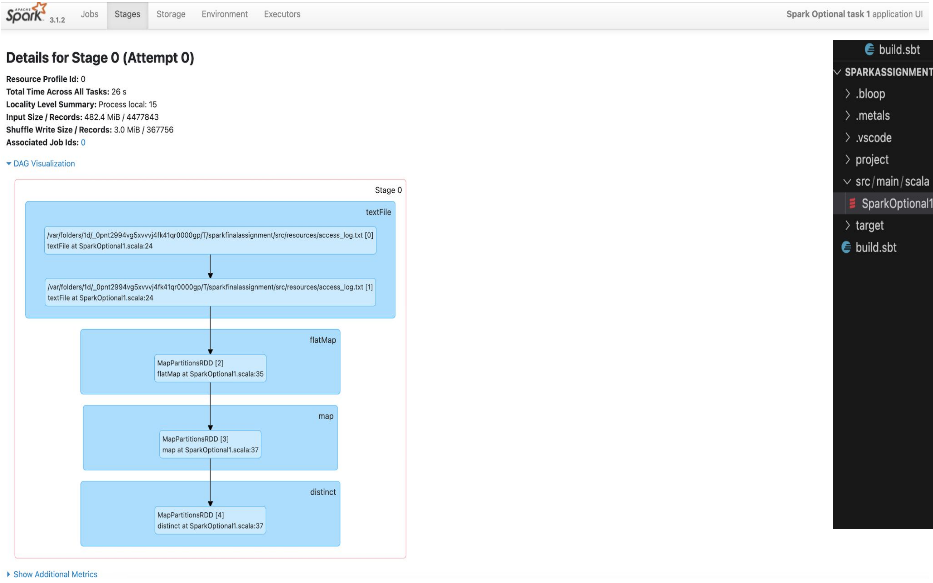
# Flow Diagram of Scenario - 3



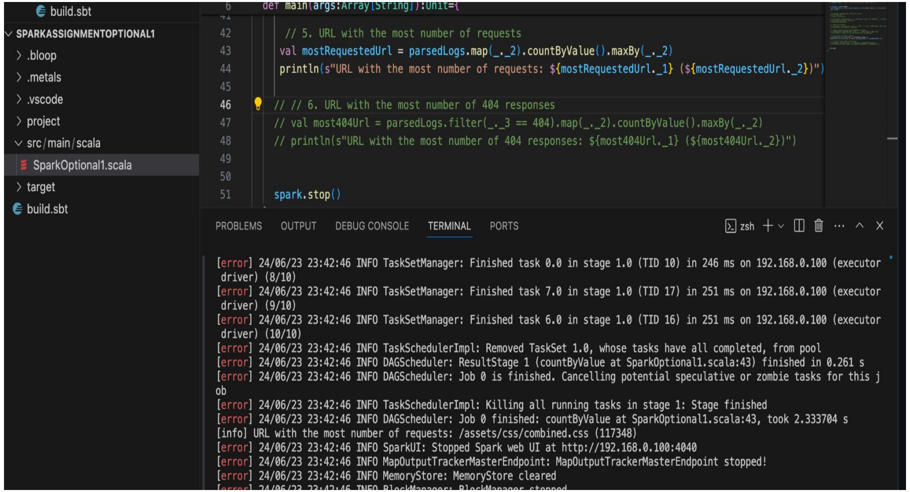
# Optional Scenario 1

**Description:** Optional Scenario 1 uses RDDs in Apache Spark to process access log data from Kaggle in text format. Steps include reading the log file, converting it to RDDs, and extracting IP addresses, URLs, response statuses, and bytes sent. Key operations include finding unique IPs, grouping URLs by 200 status, counting 4xx responses, identifying requests > 5000 bytes, determining URLs with the most requests, and finding URLs with the most 404 errors. These operations offer insights into traffic patterns, response statuses, and error occurrences in the log data. [**Code link:-** [https://github.com/r1999-ron/SparkOptional1Task](https://github.com/r1999-ron/SparkOptional1Task)]

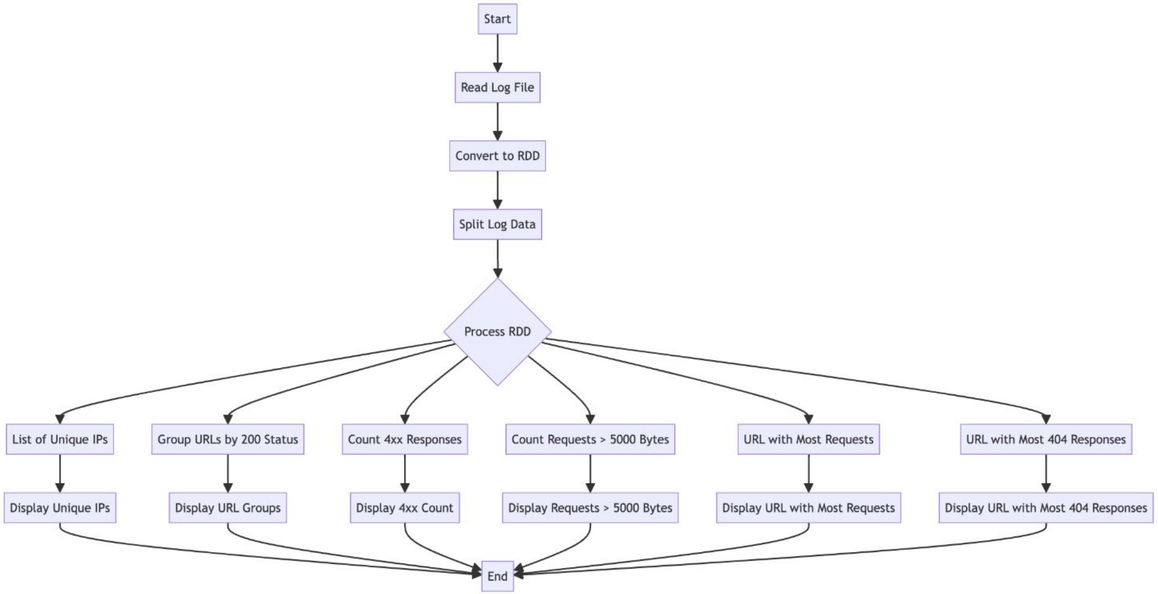
## Screenshot of DAG output




## Screenshot of spark job operation



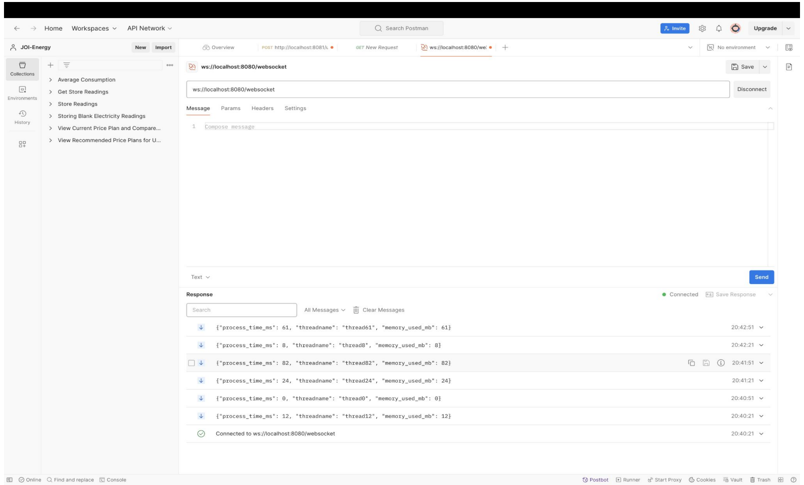
# Flow Diagram - Optional 1



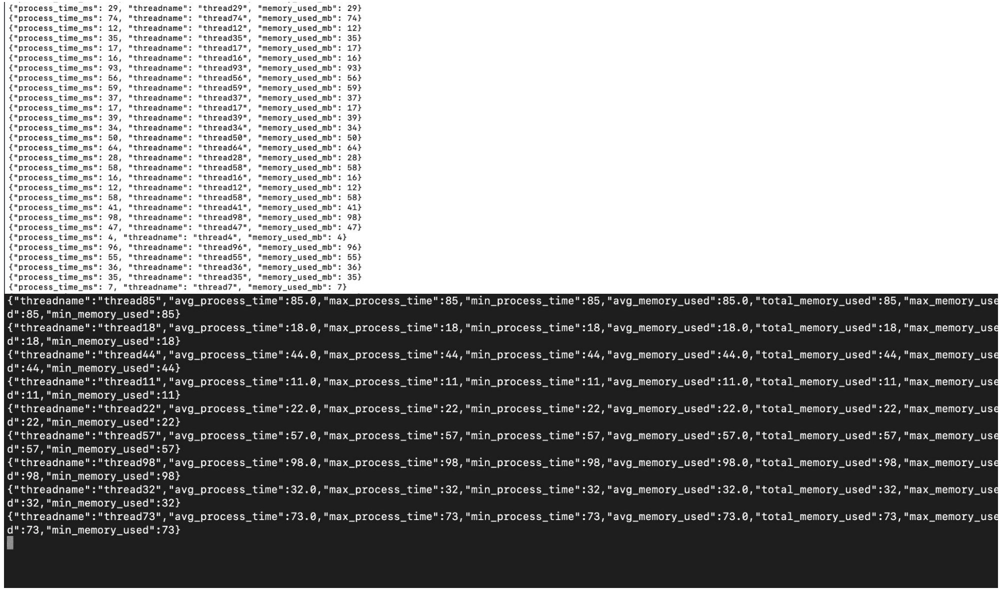
# Optional Scenario 5

**Description**: Optional Scenario 5 involves creating a WebSocket using the Akka framework to produce JSON objects every 30 seconds with process time, thread name, and memory used. A Spark Streaming job consumes this WebSocket data, writes it to a Kafka topic in AVRO format, and another Spark Streaming job performs aggregation operations on the consumed data.[**Code link:- [https://github.com/r1999-ron/SparkOptional5Task](https://github.com/r1999-ron/SparkOptional5Task)**]

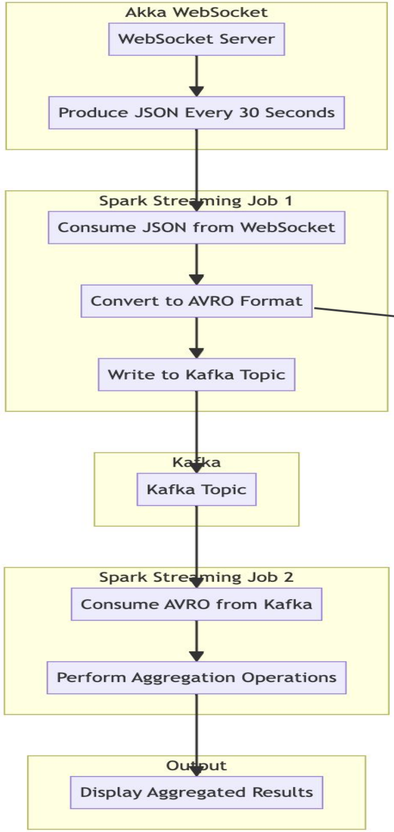
## Screenshot of postman websocket connection

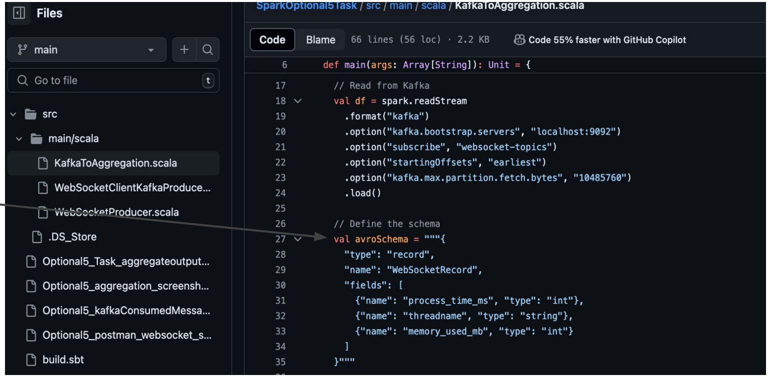



## kafka consuming websocket data & aggregation output



# Flow Diagram - Optional Task 5




```
def main(args: Array[String]): Unit = {
  // Read from Kafka
  val df = spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "localhost:9092")
    .option("subscribe", "websocket-topics")
    .option("startingOffsets", "earliest")
    .option("kafka.max.partition.fetch.bytes", "10485760")
    .load()

  // Define the schema
  val avroSchema = """{
    "type": "record",
    "name": "WebSocketRecord",
    "fields": [
      {"name": "process_time_ms", "type": "int"},
      {"name": "threadname", "type": "string"},
      {"name": "memory_used_mb", "type": "int"}
    ]
  }"""
```

# THANK YOU