

Comprehensive Data Pipeline Solutions Using Spark, Kafka, and AWS

Ronak Sengupta

Senior Software Engineer 2

Payoda Technologies

Screenshot of aggregate operations

keyspace table screenshot & DAG Output

[illegible]

The screenshot displays the Amazon Redshift console interface. At the top, the breadcrumb navigation shows the path: Amazon Keyspaces > Keyspaces > tutorialkeyspace > air_quality_data_new. The main heading is 'Table air_quality_data_new' with an 'Info' link. Below this is a 'Summary' section containing table metadata: Table name (air_quality_data_new), Keyspace name (tutorialkeyspace), Replication strategy (Single-Region), Table status (Active), Point-in-time recovery (Off), Client-side timestamp (Off), Read/write capacity settings (On-demand), Amazon Resource Name (ARN) (arn:aws:cassandra:us-east-1:992382642404:keyspace/tutorialkey), Billable table size (Unavailable), Default Time to Live (TTL = 0), and Never expire (TTL = 0). A toolbar shows navigation icons and the URL 'localhost:4040/SQL/execution/?id=2'. Below the toolbar is a tabbed interface with 'Schema', 'Capacity', 'Monitor', 'Backups', and 'Add'. The 'Schema' tab is active, showing a 'Columns (16)' section with a search bar and a table of columns: 'station_name' (Partition key, text) and 'timestamp' (timestamp). To the right, the 'Details for Query 2' section shows query execution metrics: Submitted Time (2024/06/19 16:19:35), Duration (44 s), and Running Jobs (3). A checkbox for 'Show the Stage ID and Task ID that corresponds to the max metric' is present. Below this, a diagram shows a flow from a 'Scan csv' task to an 'AppendData' task. The 'Scan csv' task box contains the following metrics: number of output rows: 22,182; number of files read: 1; metadata time: 0 ms; size of files read: 53.2 MiB.

Amazon Keyspaces > Keyspaces > tutorialkeyspace > air_quality_data_new

Table in KeySpace

Table air_quality_data_new [Info](#)

Delete

Query table

Summary

Table name air_quality_data_new	Table status Active	Read/write capacity settings Info On-demand	Billable table size Info Unavailable
Keyspace name tutorialkeyspace	Point-in-time recovery Info Off	Amazon Resource Name (ARN) arn:aws:cassandra:us-east-1:992382642404:keyspace/tutorialkey	Default Time to Live (TTL) period Info Never expire (TTL = 0)
Replication strategy Single-Region	Client-side timestamp Off		

localhost:4040/SQL/execution/?id=2

Schema

Capacity

Monitor

Backups

Add

Columns (16) [Info](#)

Find columns by column name, column type or data type

Column name	Data type
Date <div>Partition key</div>	timestamp
station_name <div>Partition key</div>	text

Details for Query 2

Submitted Time: 2024/06/19 16:19:35
Duration: 44 s
Running Jobs: 3

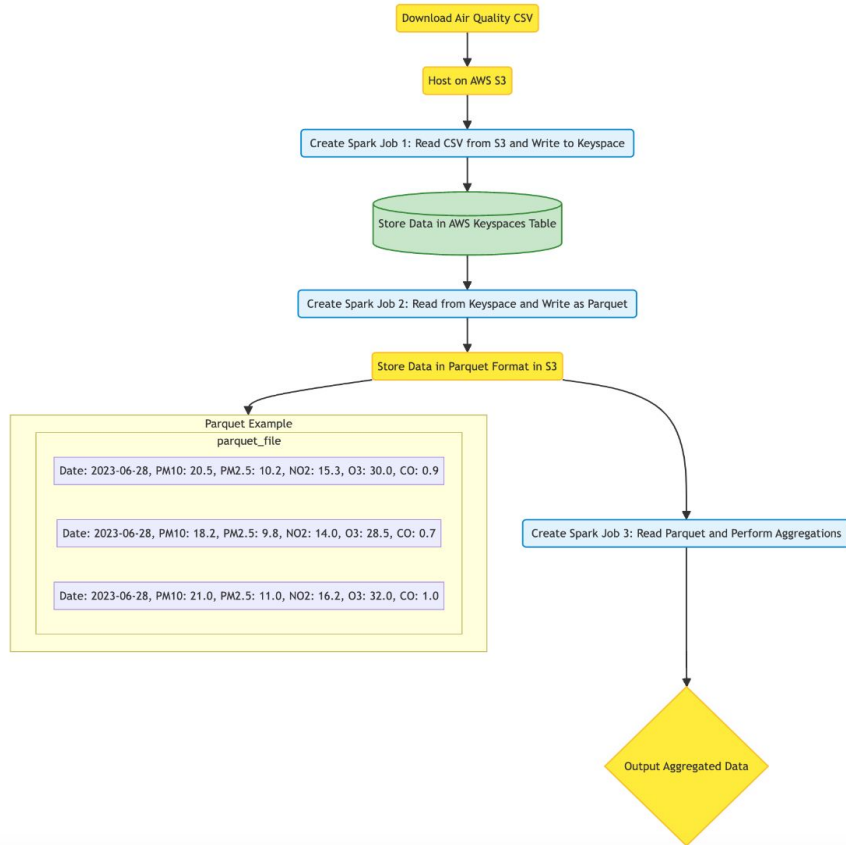
☐ Show the Stage ID and Task ID that corresponds to the max metric

Scan csv
number of output rows: 22,182
number of files read: 1
metadata time: 0 ms
size of files read: 53.2 MiB

AppendData

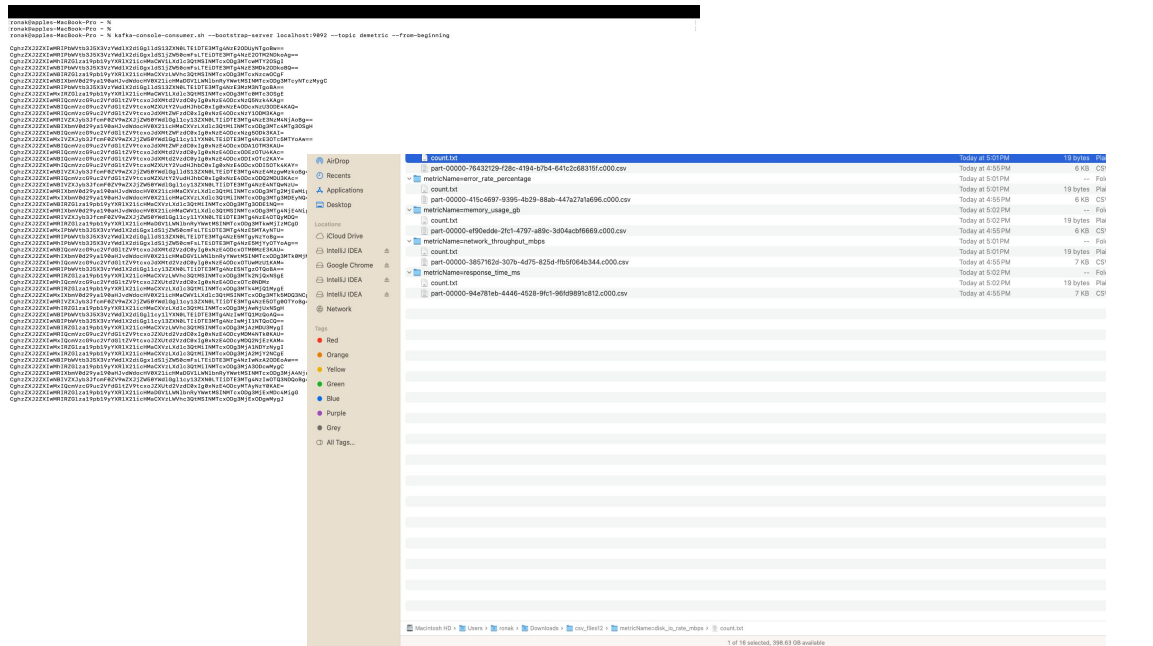
[Details](#)

Flow Diagram - Scenario 1:



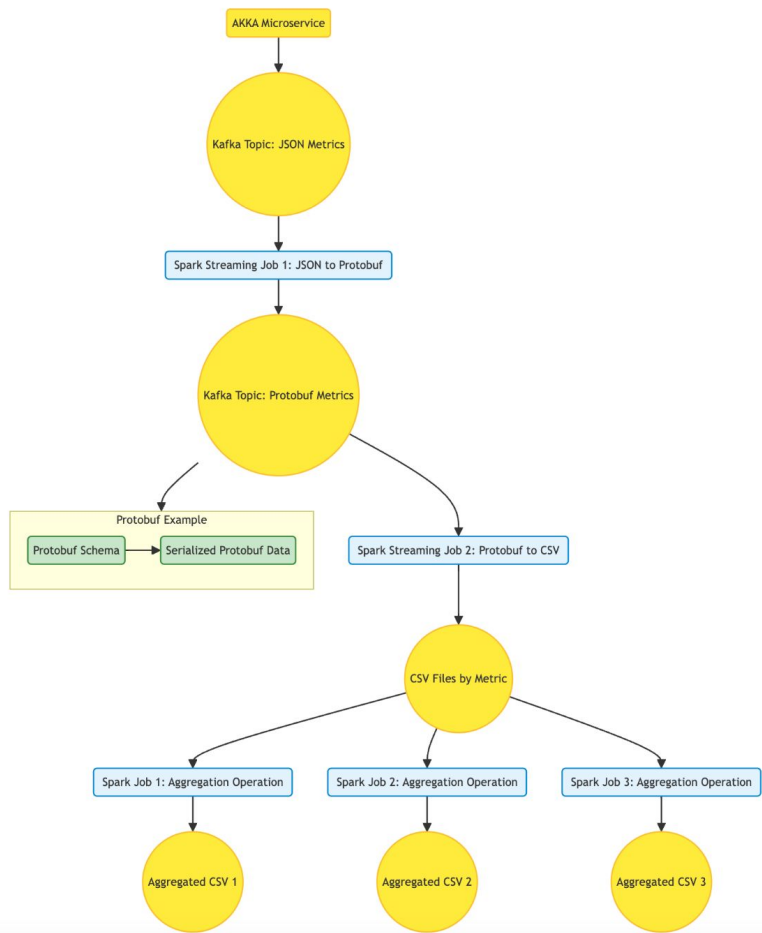
Description: In Scenario 2, a streaming pipeline using AKKA, Apache Spark, Protobuf, Spark Streaming, and Apache Kafka was implemented. An AKKA microservice generates JSON server metrics data posted to Kafka every 5 seconds. Spark Streaming converts JSON to Protobuf and publishes to another Kafka topic, 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 binary data(protobuf format) & output csv screenshots



1	server_name	total_sum	count	average
2	server02	436	8	54.5

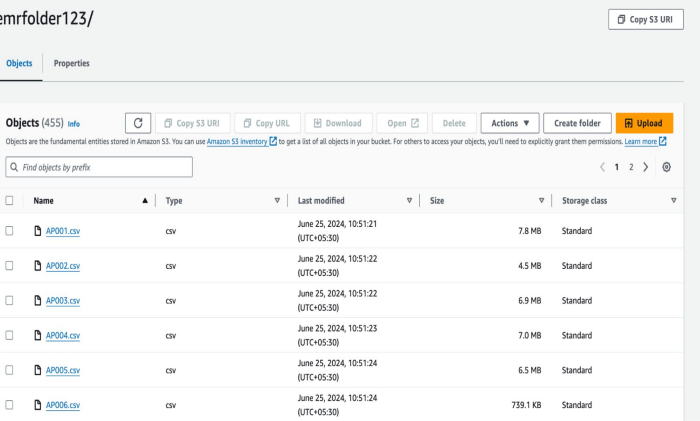
Flow Diagram - Scenario 2:



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:



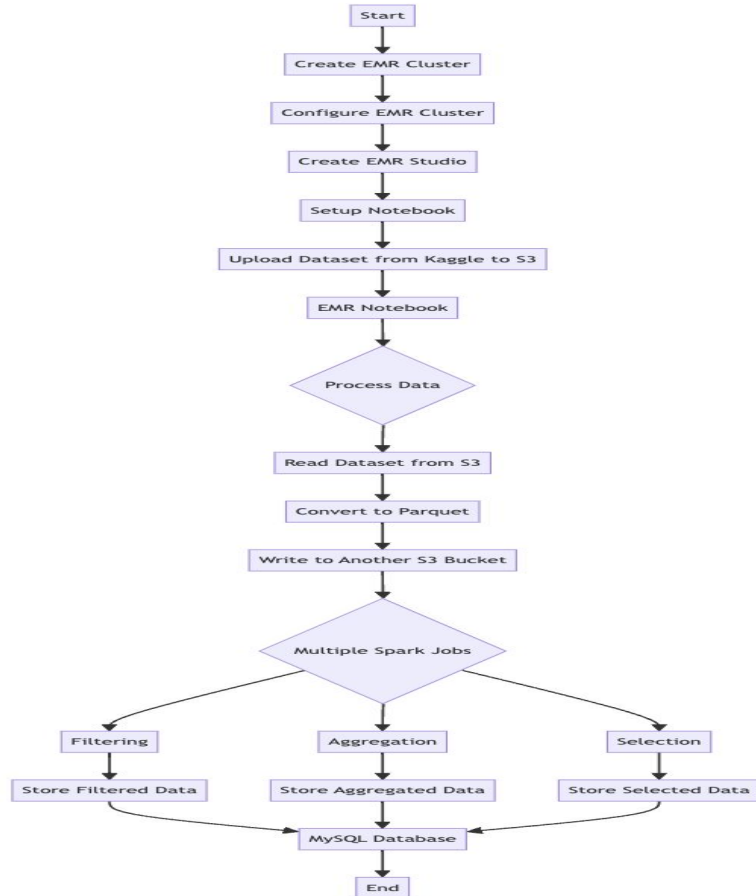
Spark Job filtration screenshot

```
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	244.7440604026846
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>]

Screenshot of DAG output



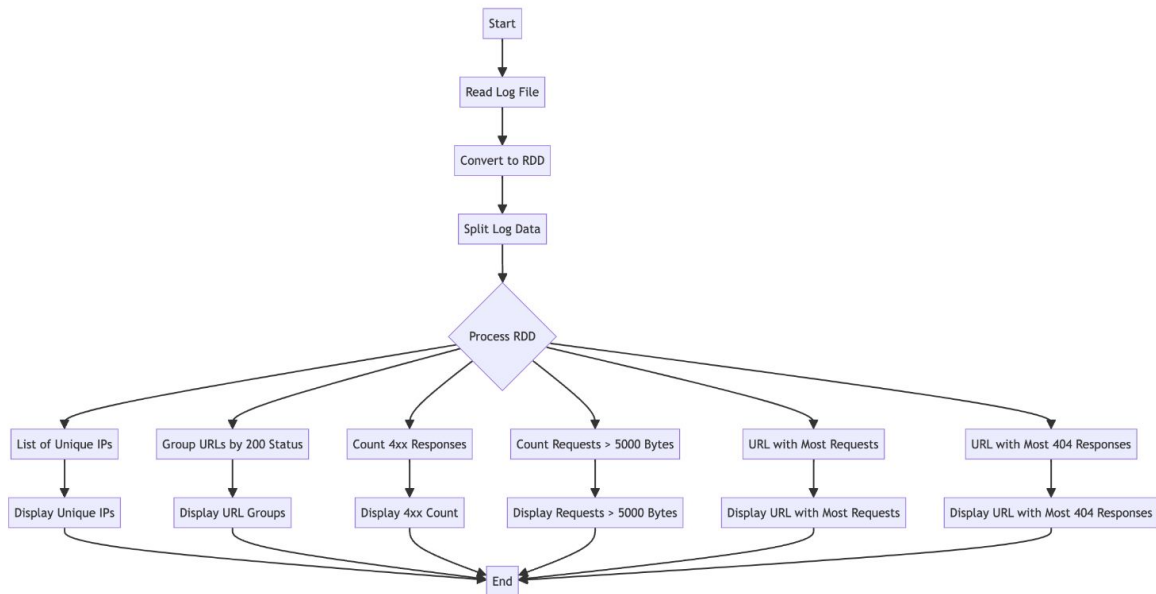
Screenshot of spark job operation

The screenshot shows the Spark job operation. The left pane displays the code for `SparkOptional1.scala`, which includes comments and Scala code for processing log data. The right pane shows the Spark web UI logs, which include error messages and status updates. The logs indicate that the job finished successfully, with a final status of `Job 0 finished: countByValue at SparkOptional1.scala:43, took 2.333704 s`.

```
1 // 5. URL with the most number of requests
2 // 3. URL with the most number of 404 responses
3 // 4. URL with the most number of 200 responses
4 // 6. URL with the most number of 404 responses
5 // 7. URL with the most number of 200 responses
6 // 8. URL with the most number of 404 responses
7 // 9. URL with the most number of 200 responses
8 // 10. URL with the most number of 404 responses
9 // 11. URL with the most number of 200 responses
10 // 12. URL with the most number of 404 responses
11 // 13. URL with the most number of 200 responses
12 // 14. URL with the most number of 404 responses
13 // 15. URL with the most number of 200 responses
14 // 16. URL with the most number of 404 responses
15 // 17. URL with the most number of 200 responses
16 // 18. URL with the most number of 404 responses
17 // 19. URL with the most number of 200 responses
18 // 20. URL with the most number of 404 responses
19 // 21. URL with the most number of 200 responses
20 // 22. URL with the most number of 404 responses
21 // 23. URL with the most number of 200 responses
22 // 24. URL with the most number of 404 responses
23 // 25. URL with the most number of 200 responses
24 // 26. URL with the most number of 404 responses
25 // 27. URL with the most number of 200 responses
26 // 28. URL with the most number of 404 responses
27 // 29. URL with the most number of 200 responses
28 // 30. URL with the most number of 404 responses
29 // 31. URL with the most number of 200 responses
30 // 32. URL with the most number of 404 responses
31 // 33. URL with the most number of 200 responses
32 // 34. URL with the most number of 404 responses
33 // 35. URL with the most number of 200 responses
34 // 36. URL with the most number of 404 responses
35 // 37. URL with the most number of 200 responses
36 // 38. URL with the most number of 404 responses
37 // 39. URL with the most number of 200 responses
38 // 40. URL with the most number of 404 responses
39 // 41. URL with the most number of 200 responses
40 // 42. URL with the most number of 404 responses
41 // 43. URL with the most number of 200 responses
42 // 44. URL with the most number of 404 responses
43 // 45. URL with the most number of 200 responses
44 // 46. URL with the most number of 404 responses
45 // 47. URL with the most number of 200 responses
46 // 48. URL with the most number of 404 responses
47 // 49. URL with the most number of 200 responses
48 // 50. URL with the most number of 404 responses
49 // 51. URL with the most number of 200 responses
50 // 52. URL with the most number of 404 responses
51 // 53. URL with the most number of 200 responses
52 // 54. URL with the most number of 404 responses
53 // 55. URL with the most number of 200 responses
54 // 56. URL with the most number of 404 responses
55 // 57. URL with the most number of 200 responses
56 // 58. URL with the most number of 404 responses
57 // 59. URL with the most number of 200 responses
58 // 60. URL with the most number of 404 responses
59 // 61. URL with the most number of 200 responses
60 // 62. URL with the most number of 404 responses
61 // 63. URL with the most number of 200 responses
62 // 64. URL with the most number of 404 responses
63 // 65. URL with the most number of 200 responses
64 // 66. URL with the most number of 404 responses
65 // 67. URL with the most number of 200 responses
66 // 68. URL with the most number of 404 responses
67 // 69. URL with the most number of 200 responses
68 // 70. URL with the most number of 404 responses
69 // 71. URL with the most number of 200 responses
70 // 72. URL with the most number of 404 responses
71 // 73. URL with the most number of 200 responses
72 // 74. URL with the most number of 404 responses
73 // 75. URL with the most number of 200 responses
74 // 76. URL with the most number of 404 responses
75 // 77. URL with the most number of 200 responses
76 // 78. URL with the most number of 404 responses
77 // 79. URL with the most number of 200 responses
78 // 80. URL with the most number of 404 responses
79 // 81. URL with the most number of 200 responses
80 // 82. URL with the most number of 404 responses
81 // 83. URL with the most number of 200 responses
82 // 84. URL with the most number of 404 responses
83 // 85. URL with the most number of 200 responses
84 // 86. URL with the most number of 404 responses
85 // 87. URL with the most number of 200 responses
86 // 88. URL with the most number of 404 responses
87 // 89. URL with the most number of 200 responses
88 // 90. URL with the most number of 404 responses
89 // 91. URL with the most number of 200 responses
90 // 92. URL with the most number of 404 responses
91 // 93. URL with the most number of 200 responses
92 // 94. URL with the most number of 404 responses
93 // 95. URL with the most number of 200 responses
94 // 96. URL with the most number of 404 responses
95 // 97. URL with the most number of 200 responses
96 // 98. URL with the most number of 404 responses
97 // 99. URL with the most number of 200 responses
100 // 100. URL with the most number of 404 responses
```

```
24/06/23 23:42:46 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 10) in 246 ms on 192.168.0.100 (executor driver) (8/10)
24/06/23 23:42:46 INFO TaskSetManager: Finished task 7.0 in stage 1.0 (TID 17) in 251 ms on 192.168.0.100 (executor driver) (9/10)
24/06/23 23:42:46 INFO TaskSetManager: Finished task 6.0 in stage 1.0 (TID 16) in 251 ms on 192.168.0.100 (executor driver) (10/10)
24/06/23 23:42:46 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
24/06/23 23:42:46 INFO DAGScheduler: ResultStage 1 (countByValue at SparkOptional1.scala:43) finished in 0.261 s
24/06/23 23:42:46 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
24/06/23 23:42:46 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
24/06/23 23:42:46 INFO DAGScheduler: Job 0 finished: countByValue at SparkOptional1.scala:43, took 2.333704 s
[info] URL with the most number of requests: /assets/css/combined.css (117348)
24/06/23 23:42:46 INFO SparkUI: Stopped Spark web UI at http://192.168.0.100:4040
24/06/23 23:42:46 INFO MemoryStore: MemoryStore cleared
24/06/23 23:42:46 INFO BlockManager: BlockManager stopped
```

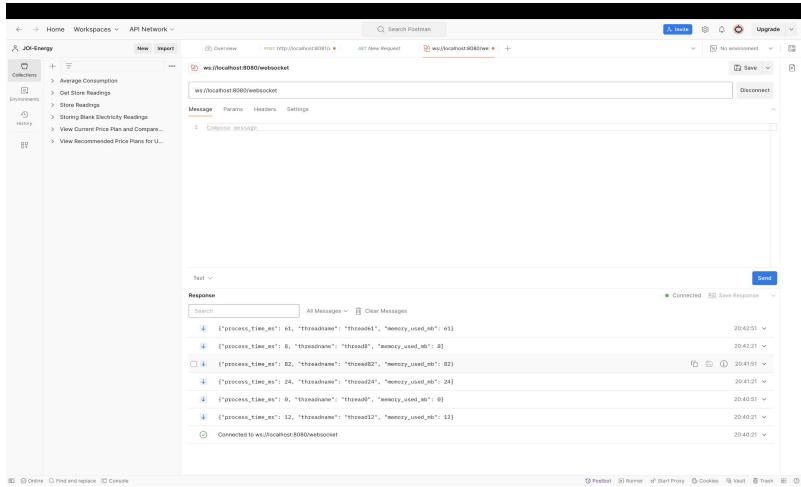

Flow Diagram - Optional 1



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)]

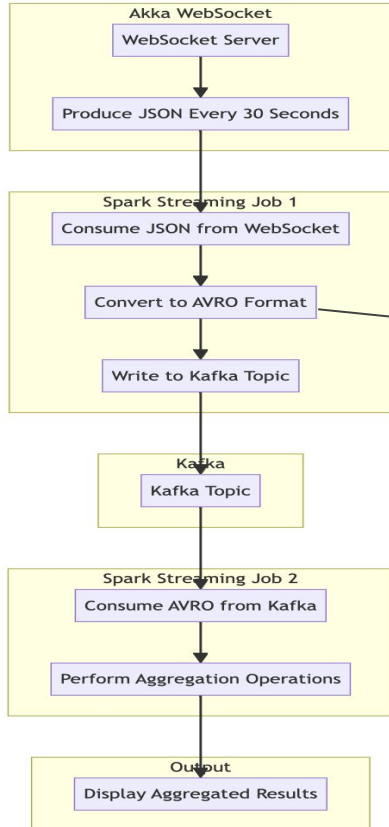
Screenshot of postman websocket connection



kafka consuming websocket data & aggregation output

```
{
  "process_time_ms": 29,
  "threadname": "thread29",
  "memory_used_mb": 293
}
{"process_time_ms": 74, "threadname": "thread74", "memory_used_mb": 743}
{"process_time_ms": 12, "threadname": "thread12", "memory_used_mb": 123}
{"process_time_ms": 35, "threadname": "thread35", "memory_used_mb": 353}
{"process_time_ms": 17, "threadname": "thread17", "memory_used_mb": 173}
{"process_time_ms": 16, "threadname": "thread16", "memory_used_mb": 163}
{"process_time_ms": 93, "threadname": "thread93", "memory_used_mb": 933}
{"process_time_ms": 56, "threadname": "thread56", "memory_used_mb": 563}
{"process_time_ms": 59, "threadname": "thread59", "memory_used_mb": 593}
{"process_time_ms": 37, "threadname": "thread37", "memory_used_mb": 373}
{"process_time_ms": 17, "threadname": "thread17", "memory_used_mb": 173}
{"process_time_ms": 39, "threadname": "thread39", "memory_used_mb": 393}
{"process_time_ms": 34, "threadname": "thread34", "memory_used_mb": 343}
{"process_time_ms": 58, "threadname": "thread58", "memory_used_mb": 583}
{"process_time_ms": 64, "threadname": "thread64", "memory_used_mb": 643}
{"process_time_ms": 28, "threadname": "thread28", "memory_used_mb": 283}
{"process_time_ms": 58, "threadname": "thread58", "memory_used_mb": 583}
{"process_time_ms": 16, "threadname": "thread16", "memory_used_mb": 163}
{"process_time_ms": 12, "threadname": "thread12", "memory_used_mb": 123}
{"process_time_ms": 58, "threadname": "thread58", "memory_used_mb": 583}
{"process_time_ms": 41, "threadname": "thread41", "memory_used_mb": 413}
{"process_time_ms": 98, "threadname": "thread98", "memory_used_mb": 983}
{"process_time_ms": 47, "threadname": "thread47", "memory_used_mb": 473}
{"process_time_ms": 4, "threadname": "thread4", "memory_used_mb": 43}
{"process_time_ms": 96, "threadname": "thread96", "memory_used_mb": 963}
{"process_time_ms": 65, "threadname": "thread65", "memory_used_mb": 653}
{"process_time_ms": 36, "threadname": "thread36", "memory_used_mb": 363}
{"process_time_ms": 35, "threadname": "thread35", "memory_used_mb": 353}
{"process_time_ms": 7, "threadname": "thread7", "memory_used_mb": 73}
}
{"threadname": "thread85", "avg_process_time": 85.0, "max_process_time": 85, "min_process_time": 85, "avg_memory_used": 85.0, "total_memory_used": 85, "max_memory_used": 85, "min_memory_used": 85}
{"threadname": "thread18", "avg_process_time": 18.0, "max_process_time": 18, "min_process_time": 18, "avg_memory_used": 18.0, "total_memory_used": 18, "max_memory_used": 18, "min_memory_used": 18}
{"threadname": "thread44", "avg_process_time": 44.0, "max_process_time": 44, "min_process_time": 44, "avg_memory_used": 44.0, "total_memory_used": 44, "max_memory_used": 44, "min_memory_used": 44}
{"threadname": "thread11", "avg_process_time": 11.0, "max_process_time": 11, "min_process_time": 11, "avg_memory_used": 11.0, "total_memory_used": 11, "max_memory_used": 11, "min_memory_used": 11}
{"threadname": "thread22", "avg_process_time": 22.0, "max_process_time": 22, "min_process_time": 22, "avg_memory_used": 22.0, "total_memory_used": 22, "max_memory_used": 22, "min_memory_used": 22}
{"threadname": "thread57", "avg_process_time": 57.0, "max_process_time": 57, "min_process_time": 57, "avg_memory_used": 57.0, "total_memory_used": 57, "max_memory_used": 57, "min_memory_used": 57}
{"threadname": "thread44", "avg_process_time": 44.0, "max_process_time": 44, "min_process_time": 44, "avg_memory_used": 44.0, "total_memory_used": 44, "max_memory_used": 44, "min_memory_used": 44}
{"threadname": "thread98", "avg_process_time": 98.0, "max_process_time": 98, "min_process_time": 98, "avg_memory_used": 98.0, "total_memory_used": 98, "max_memory_used": 98, "min_memory_used": 98}
{"threadname": "thread32", "avg_process_time": 32.0, "max_process_time": 32, "min_process_time": 32, "avg_memory_used": 32.0, "total_memory_used": 32, "max_memory_used": 32, "min_memory_used": 32}
{"threadname": "thread73", "avg_process_time": 73.0, "max_process_time": 73, "min_process_time": 73, "avg_memory_used": 73.0, "total_memory_used": 73, "max_memory_used": 73, "min_memory_used": 73}
}
```

Flow Diagram - Optional Task 5



The screenshot shows a code editor with the file `KafkaToAggregation.scala` open. The code defines a `main` function that reads a stream from Kafka, converts it to AVRO format, and writes it to a Kafka topic. The schema for the AVRO data is defined as follows:

```
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