Project Documentation: Real-Time IoT Data Stream Processing and Aggregation System

Overview

This project consists of three primary components:

1. **IoT Sensor Module**: A real-time IoT module written in Scala with Akka that produces sensor readings and pushes them to a Kafka topic at a rate of 10 ms per message.
2. **Spark Jobs Application**: A Spark application that consumes the sensor readings from Kafka using Spark Streaming, aggregates them, stores them in a Google Cloud Storage (GCS) bucket in Protobuf format, and uploads an aggregated JSON file.
3. **Akka HTTP Application**: An Akka HTTP service that exposes endpoints for querying the aggregated sensor readings.

Key Technologies

- **Scala** and **Akka** for the IoT sensor module.
- **Kafka** for message streaming.
- **Apache Spark** for stream processing and aggregation.
- **Google Cloud Storage (GCS)** for data storage.
- **Protobuf** for efficient binary data serialization.
- **Akka HTTP** for exposing RESTful APIs.

1. IoT Sensor Module (Akka Producer)

Purpose

The IoT module generates sensor readings and sends them to a Kafka topic in real-time. The system is designed to produce messages at a very high frequency (every 10 milliseconds).

Components:

- **Akka Actors**: Used for concurrent, non-blocking operations to handle the high frequency of message production.
- **Kafka Producer**: Pushes the sensor readings to a Kafka topic.

Data Flow:

- Sensor readings (e.g., temperature, humidity, pressure, etc.) are generated at 10 ms intervals.
- These readings are encapsulated in a message and sent to the Kafka topic.

Kafka Topic Configuration:

- **Topic Name**: `sensor-readings`
- **Partitioning**: Based on sensor ID or timestamp to allow for parallelism.

Example Message Structure:

```
message SensorReading {
  string sensorId = 1;
  int64 timestamp = 2;
  float temperature = 3;
  float humidity = 4;
}
```

Configuration:

- **Akka Dispatcher**: To handle high-throughput asynchronous tasks.
- **Kafka Producer Settings**: Configure Kafka producer for high-throughput publishing.

2. Spark Jobs Application (Stream Processing and Aggregation)

Purpose

The Spark application consumes the sensor readings from the Kafka topic in real-time, performs necessary aggregations, and stores the results in a GCS bucket partitioned by date and hour. Additionally, the aggregated data is uploaded as a JSON file for easy access.

Components:

- **Spark Streaming**: To consume real-time data from Kafka.
- **GCS Sink**: To store aggregated data in a partitioned format (Protobuf) and upload JSON files with the aggregated data.
- **Aggregation Logic**: Aggregates the sensor data in batches triggered every 5 seconds.

Data Flow:

- **Batch Interval**: Spark streaming processes data in 5-second intervals.
- **Aggregation**: Data is aggregated over time by calculating statistics like average temperature, humidity, etc., over the past 5-second window.
- **Partitioning**: The aggregated data is written to GCS in a partitioned directory format: `YYYY/MM/DD/HH/`.
- **Protobuf Format**: Data is stored in Protobuf format for efficient storage.
- **JSON File**: The aggregated data is also stored as a JSON file for easy access and analysis.

GCS Directory Structure:

```
gs://my-bucket/sensor-data/YYYY/MM/DD/HH/
```

Batch Processing Example:

- **Batch Interval**: 5 seconds.
- **Aggregation**: For each batch, calculate the mean or sum of sensor readings over the past 5 seconds.

Sample Aggregated Data (Protobuf):

```
message AggrData {
  string sensorId = 1;           // Unique identifier for the
sensor
  float avgTemperature = 2;      // Average temperature
  float avgHumidity = 3;         // Average humidity
  float minTemperature = 4;      // Minimum temperature
  float maxTemperature = 5;      // Maximum temperature
  float minHumidity = 6;         // Minimum humidity
  float maxHumidity = 7;         // Maximum humidity
  int32 dataCount = 8;        // Count of data points which w
ere combined in the aggregate
}
```

Example JSON File (Aggregated Data):

```
{
        "sensorId": "05a4a3b8-a366-4232-92c6-bda62372a46c",
        "avgTemperature": 31.104897,
        "avgHumidity": 17.83693,
        "minTemperature": 0.0,
        "maxTemperature": 31.104897,
        "minHumidity": 0.0,
        "maxHumidity": 17.83693,
        "dataCount": 2
    }
```

Spark Application Configuration:

- **Kafka Source**: Reading from Kafka topic `sensor-readings`.

- **Windowing**: Using a 5-second window for aggregations.
- **Protobuf and JSON Formats**: Writing data to GCS in Protobuf format and uploading a JSON summary.

3. Akka HTTP Application (API Service)

Purpose

The Akka HTTP service exposes endpoints that allow users to query the aggregated sensor readings stored in GCS.

Components:

- **Akka HTTP**: Handles HTTP requests and provides endpoints for querying the aggregated data.
- **GCS Access**: The service fetches the aggregated data stored in GCS.
- **JSON Response**: Aggregated data is returned as a JSON response for easy consumption by clients.

API Endpoints:

1. Get Aggregated Data

- **URL**: `/aggregated-data/sensorId`
- **Method**: `GET`
- **Response**: Returns aggregated sensor readings in JSON format.

Example Request:

```
GET /aggregated-data/05a4a3b8-a366-4232-92c6-bda62372a46c
```

Example Response:

```
{
        "sensorId": "05a4a3b8-a366-4232-92c6-bda62372a46c",
        "avgTemperature": 31.104897,
```

```
        "avgHumidity": 17.83693,
        "minTemperature": 0.0,
        "maxTemperature": 31.104897,
        "minHumidity": 0.0,
        "maxHumidity": 17.83693,
        "dataCount": 2
    }
```

2. Get All Sensor Aggregated Data

- **URL**: `/aggregated-data`
- **Method**: `GET`
- **Response**: Returns all aggregated data for all sensors.

Example Request:

```
GET /aggregated-data
```

Example Response:

```
[
  {
        "sensorId": "05a4a3b8-a366-4232-92c6-bda62372a46c",
        "avgTemperature": 31.104897,
        "avgHumidity": 17.83693,
        "minTemperature": 0.0,
        "maxTemperature": 31.104897,
        "minHumidity": 0.0,
        "maxHumidity": 17.83693,
        "dataCount": 2
    },
    {
        "sensorId": "06e4fc3a-720b-42d2-b4ee-3eee8271ed00",
        "avgTemperature": -0.61128235,
        "avgHumidity": 23.738659,
```

```
        "minTemperature": -0.61128235,
        "maxTemperature": 0.0,
        "minHumidity": 0.0,
        "maxHumidity": 23.738659,
        "dataCount": 2
    },
]
```

Akka HTTP Server Configuration:

- **Route Definitions**: Define routes to handle HTTP requests and process them accordingly.
- **GCS Access**: Use Google Cloud SDK to fetch the aggregated data from GCS.
- **Serialization**: JSON is used to return aggregated data in a readable format.

System Workflow

Step-by-Step Process:

1. **IoT Sensor Module** generates sensor data every 10 ms and sends it to Kafka.
2. **Spark Jobs Application** consumes the data from Kafka in 5-second batches, aggregates it, and stores it in GCS in Protobuf format. It also generates a JSON file for easy access.
3. **Akka HTTP Application** provides a RESTful API to retrieve aggregated sensor data stored in GCS.

System Configuration

Kafka:

- **Topic**: `sensor-readings`
- **Partitioning Key**: `sensor_id` or `timestamp`

Spark Streaming:

- **Batch Interval**: 5 seconds

- **GCS raw data Path**: `gs://de_case_study_bucket/raw/sensor-data/YYYY/MM/DD/HH/`
- **GCS aggregated data path:**
  - **Protobuf**: `gs://de_case_study_bucket/aggregated/protobuf/sensor-data/YYYY/MM/DD/HH/`
  - **JSON**: `gs://de_case_study_bucket/aggregated/json/sensor-data/YYYY/MM/DD/HH/`
- **File Format**: Protobuf for storage, JSON for uploading

IOT Backend:

- **API Routes**: `/aggregated-data`, `/aggregated-data/sensorId`
- **Data Fetching**: Retrieves aggregated data from GCS and serves it in JSON format.

IOT Frontend:

- UI built using react and bootstrap to consume the IOT-Backend API's and show the aggregated sensor metrics to the user

Monitoring and Scaling

Monitoring:

- **Akka**: Use Akka Monitoring and Logging to track message production and consumption rates.
- **Spark**: Use Spark UI to monitor streaming jobs and performance.
- **GCS**: Monitor storage usage to ensure that data is written correctly.

Scaling:

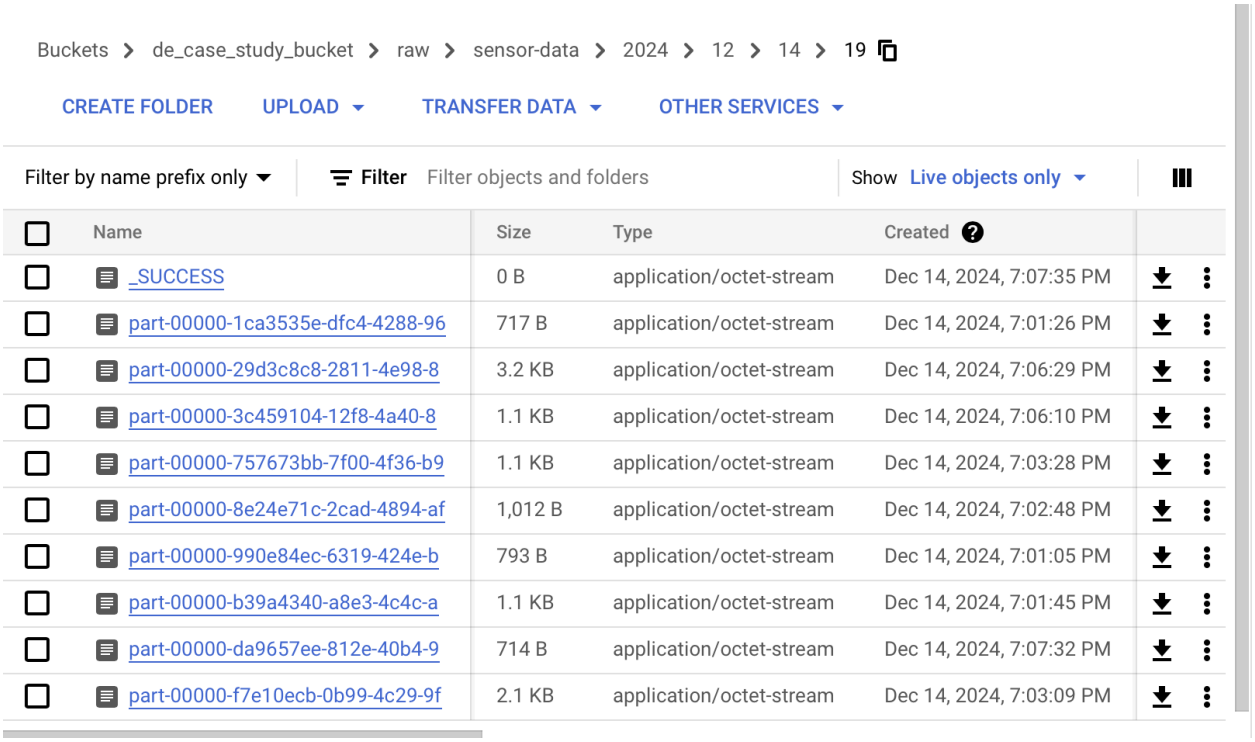
- **Kafka**: Scale Kafka brokers to handle higher throughput.
- **Spark**: Scale the number of executors and workers to handle increased batch data.
- **Akka HTTP**: Horizontal scaling of Akka HTTP servers for better API handling.

Results:

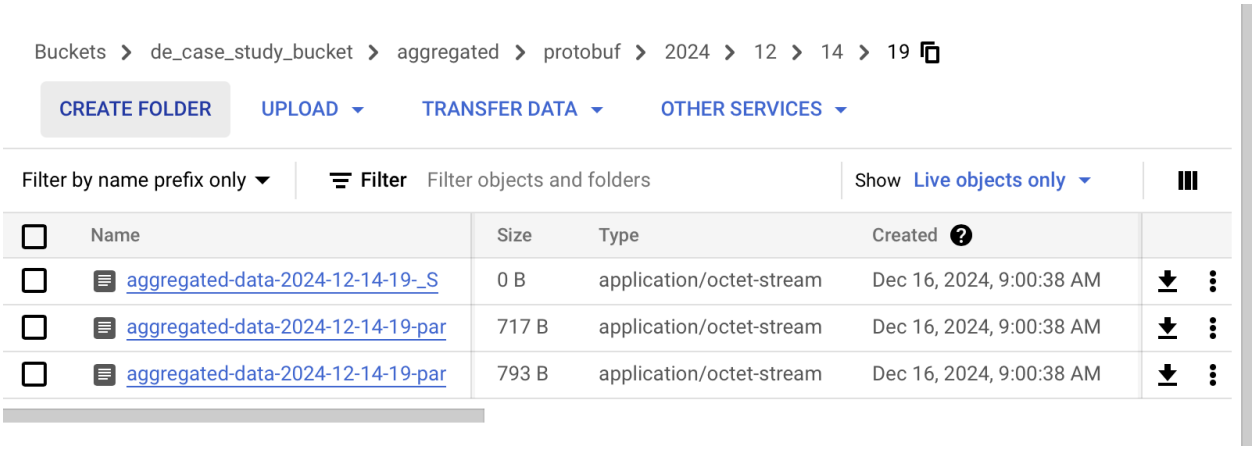
Sensor Readings:

```
{"humidity":22.707874,"sensorId":"cf0e4352-6156-4fc6-86e6-c1aa23ed933a","temperature":2.0105133,"timestamp":1734183436645}
{"humidity":10.180944,"sensorId":"3efaa768-fd6e-41e7-8684-de7ffa0ccef5","temperature":135.7703,"timestamp":1734183437665}
{"humidity":98.717186,"sensorId":"a57dfe74-e763-40f2-89bc-1c47c9cf721d","temperature":-8.026146,"timestamp":1734183438676}
{"humidity":79.826935,"sensorId":"3821cca3-c460-4a60-94fe-ba9e6a3d1de5","temperature":-21.957756,"timestamp":1734183439695}
{"humidity":85.03808,"sensorId":"75abb192-db7f-408e-a87c-1961a5651603","temperature":-36.596725,"timestamp":1734183440716}
{"humidity":31.451744,"sensorId":"f182b9c1-f86d-48a0-95d4-f6ebaaea13a7","temperature":-12.159504,"timestamp":1734183441725}
{"humidity":66.93189,"sensorId":"6e5eae9a-9e06-48ec-8f52-d381fc2384fd","temperature":-49.26902,"timestamp":1734183442746}
{"humidity":66.45687,"sensorId":"8580deda-23c9-4433-afb7-730b3f9d5eb4","temperature":-10.896278,"timestamp":1734183443765}
{"humidity":54.52777,"sensorId":"3829fc82-0b89-47a9-8221-2f2ad5689043","temperature":35.345306,"timestamp":1734183444775}
{"humidity":85.421196,"sensorId":"805ac8d7-3438-4d02-a11a-c70c6ed6e497","temperature":103.37184,"timestamp":1734183445795}
{"humidity":92.02029,"sensorId":"8a792bbb-af17-4048-9601-736fca17fbdd","temperature":-4.0955544,"timestamp":1734183446805}
{"humidity":39.443443,"sensorId":"a672e79a-4aba-4af9-a97a-f970449ce61d","temperature":144.93633,"timestamp":1734183447825}
{"humidity":48.60351,"sensorId":"1336f53b-493f-4988-afdc-1fad21800f0a","temperature":27.035698,"timestamp":1734183448835}
{"humidity":93.45907,"sensorId":"a52154b4-5107-4e98-bc65-c54e2deb9717","temperature":106.9621,"timestamp":1734183449856}
{"humidity":80.19545,"sensorId":"1304e31a-b9d1-4e95-b9c7-543141ab6005","temperature":91.42699,"timestamp":1734183450875}
{"humidity":80.283424,"sensorId":"8c89c53c-7ebc-4b62-94d1-20afa83a4ece","temperature":-18.094276,"timestamp":1734183451894}
{"humidity":86.45948,"sensorId":"7ad8116b-7fc5-4773-a7c2-5f5c268e2c1b","temperature":65.151665,"timestamp":1734183452915}
{"humidity":5.4511013,"sensorId":"2014612f-8fd1-4294-b463-7484266726a","temperature":-7.4706306,"timestamp":1734183453934}
{"humidity":98.82047,"sensorId":"c8e46f9e-813d-4984-a180-b04f410b976e","temperature":114.45915,"timestamp":1734183454955}
{"humidity":87.40871,"sensorId":"67c80cca-d41d-48ef-aec4-14a2afa25529","temperature":22.533585,"timestamp":1734183455965}
{"humidity":94.85708,"sensorId":"96eb8474-fd4b-4713-8fe6-40be4c9f5fc5","temperature":-49.499428,"timestamp":1734183456983}
{"humidity":74.43654,"sensorId":"ffca7d84-6f88-42cc-9bac-faba1091c391","temperature":-11.344383,"timestamp":1734183458006}
{"humidity":54.44395,"sensorId":"dfee8ad2-efb1-484a-b279-0554e98f2f7d","temperature":83.457596,"timestamp":1734183459026}
{"humidity":59.545433,"sensorId":"28aa2de4-57f8-41da-a91b-59f5be1eeec8","temperature":-35.01973,"timestamp":1734183460045}
{"humidity":29.19789,"sensorId":"074765d7-1a69-4e2d-91a5-83ee8ace38fc","temperature":16.838242,"timestamp":1734183461065}
{"humidity":34.80429,"sensorId":"5ded6883-cd95-4a5f-a301-fe51eebc7661","temperature":96.78882,"timestamp":1734183462075}
{"humidity":54.09137,"sensorId":"16521815-2b7b-4c0a-b9ff-280e28a43552","temperature":-45.535076,"timestamp":1734183463096}
{"humidity":55.948704,"sensorId":"67f0420e-3483-4e70-a3d9-110f4d79657d","temperature":-17.245125,"timestamp":1734183464113}
{"humidity":67.539085,"sensorId":"fa4d802b-a124-4697-8655-bde00dcc977d","temperature":145.6925,"timestamp":1734183465125}
{"humidity":72.11197,"sensorId":"83b923e8-4dbf-40b4-90eb-06081299f52b","temperature":-16.464497,"timestamp":1734183466145}
{"humidity":41.90522,"sensorId":"62984c3e-d541-4a06-83f6-eff3b20c4504","temperature":68.07314,"timestamp":1734183467165}
{"humidity":61.07908,"sensorId":"7c0a8507-4b00-415d-9b3c-92e4a5e09bc8","temperature":70.27893,"timestamp":1734183468185}
{"humidity":42.34925,"sensorId":"ada4f948-b1eb-4f57-bbf1-46b13355f6eb","temperature":-18.929756,"timestamp":1734183469195}
{"humidity":71.127785,"sensorId":"6b74a6c5-4565-4ac2-a524-aed9b393a8df","temperature":-37.62331,"timestamp":1734183470215}
{"humidity":99.41815,"sensorId":"94c9c09b-24f0-4a41-84eb-90c2cc2412e0","temperature":110.177795,"timestamp":1734183471226}
{"humidity":31.385237,"sensorId":"7c2b551d-70d6-41c8-a2d9-b1d6fcb73ea7","temperature":39.656464,"timestamp":1734183472246}
{"humidity":85.056526,"sensorId":"ef05d6aa-5919-49d5-a21f-e3f325ab46f6","temperature":9.215118,"timestamp":1734183473265}
{"humidity":55.35499,"sensorId":"684cf8b6-85c4-4aa9-9360-54936a8149b2","temperature":135.49243,"timestamp":1734183474285}
{"humidity":47.05998,"sensorId":"dccc0a19-6a01-4586-b016-e3a59a7e217d","temperature":-38.44739,"timestamp":1734183475305}
```

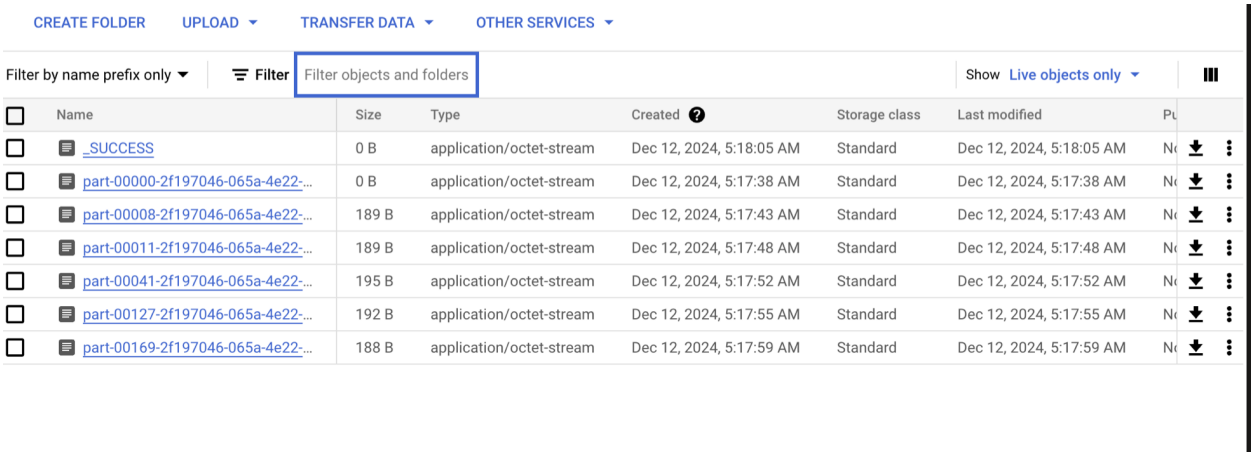

Raw Sensor Data in GCS:



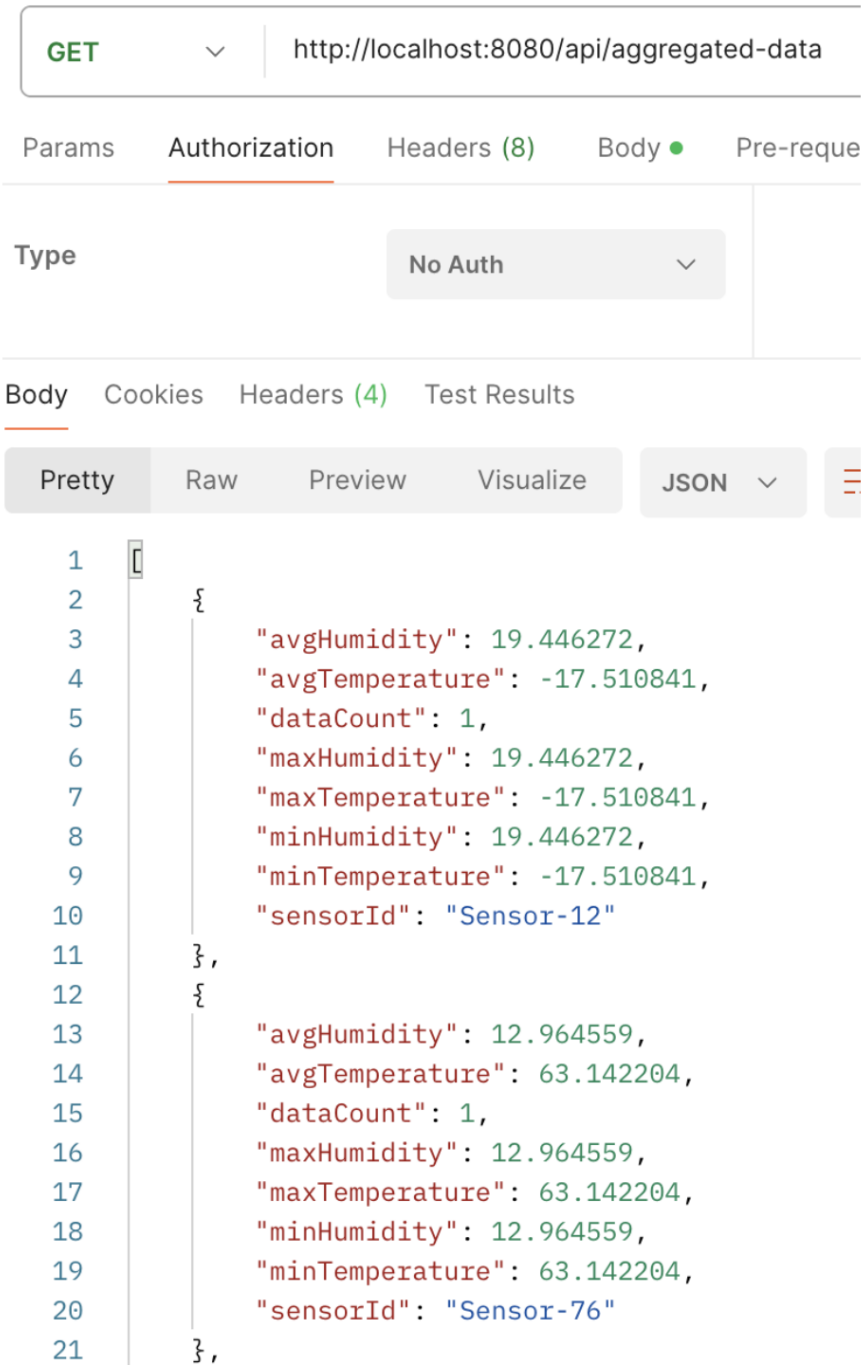
Aggregated Data in GCS:



JSON Stored in GCS:



Postman Results:


GET
http://localhost:8080/api/aggregated-data
Params
Authorization
Headers (8)
Body
Pre-reque
Type
No Auth
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"avgHumidity": 19.446272,
"avgTemperature": -17.510841,
"dataCount": 1,
"maxHumidity": 19.446272,
"maxTemperature": -17.510841,
"minHumidity": 19.446272,
"minTemperature": -17.510841,
"sensorId": "Sensor-12"
},
{
"avgHumidity": 12.964559,
"avgTemperature": 63.142204,
"dataCount": 1,
"maxHumidity": 12.964559,
"maxTemperature": 63.142204,
"minHumidity": 12.964559,
"minTemperature": 63.142204,
"sensorId": "Sensor-76"
},

Frontend UI:

Sensor Data Viewer

Search by Sensor ID

Enter Sensor ID

Search Reset

Sensor ID	Avg Temperature	Avg Humidity	Min Temperature	Max Temperature	Min Humidity	Max Humidity
134feb3c-96ce-4525-8543-32a78907052a			-3.885971	0	0	83.459274
261636fc-a79b-40e7-a975-61815ccd6ee7			0	24.903633	0	21.394085
31c02e65-761a-4919-a523-ef4173eca3e1			-6.2713623	0	0	86.15527
4f1c3c61-d8c4-453e-bfb8-4c48292eff73			0	28.16954	0	54.771263
5bcf9a08-f574-47fe-8240-6fc8539507d9	43.901825	11.344301	0	43.901825	0	11.344301
5c4e60c4-961f-418b-9541-fc20f95a4cbc	-4.8493156	28.911459	-4.8493156	0	0	28.911459
637010bb-8dda-4702-aee1-faeec32720c1	42.080048	9.047365	0	42.080048	0	9.047365
92109c72-a5ee-438f-b342-ab0586a38f65	-4.316448	54.365383	-4.316448	0	0	54.365383
a61aa5d9-6d16-470d-95a4-4e85d335eebc			0	33.899628	0	82.80055
b041f19c-91dd-48e1-b64c-daabf28d83dc			0	10.671593	0	93.59288
bcac4a54-52bd-45d1-a77f-b5413bc7e4e4	-4.203762	83.19762	-4.203762	0	0	83.19762
bea7a853-ac98-4f8f-9f30-0e6ff7582d27			0	42.52317	0	93.84731
c2a8fd25-49d8-4c43-a51c-3e0db8aa3c66	10.079765	91.87039	0	10.079765	0	91.87039
c4208e53-b717-4a42-80a9-be889e022848			0	12.630642	0	68.212685
c8b086d9-be38-4c0d-92ba-5f5608a2c69d	-5.1900024	6.9363475	-5.1900024	0	0	6.9363475
c9885cf0-8a17-4469-a58e-c9937a2be347			0	16.543068	0	23.093473
09c71129-5b49-4b12-a08a-930ef83cff08			0	28.973686	0	66.10815

Conclusion

This system provides a robust pipeline for processing and aggregating IoT sensor data in real-time. Using Akka for message production, Spark for stream processing, and Akka HTTP for exposing the data, it ensures high-throughput data handling, efficient storage in GCS, and easy access via RESTful APIs.