FINAL PROJECTS

- Real-Time IoT Data Processing and Aggregation with Visualization
- Optimized Real-Time Data Processing with Caching, Persisting, and Broadcasting

Presented By
SAKETH MUTHOJU

CASE STUDY

REAL-TIME IOT DATA PROCESSING AND AGGREGATION WITH VISUALIZATION

A scalable and robust real-time IoT data processing pipeline that seamlessly ingests, validates, and transforms high-frequency sensor readings into actionable insights through efficient storage, dynamic aggregation, and intuitive dashboard visualization.

OBJECTIVES AND
KEY FEATURES

Objectives

- Build an end-to-end IoT data pipeline with real-time processing and efficient storage.
- Ensure incremental updates for dashboard visualization.
- Maintain data accuracy and implement retention policies.

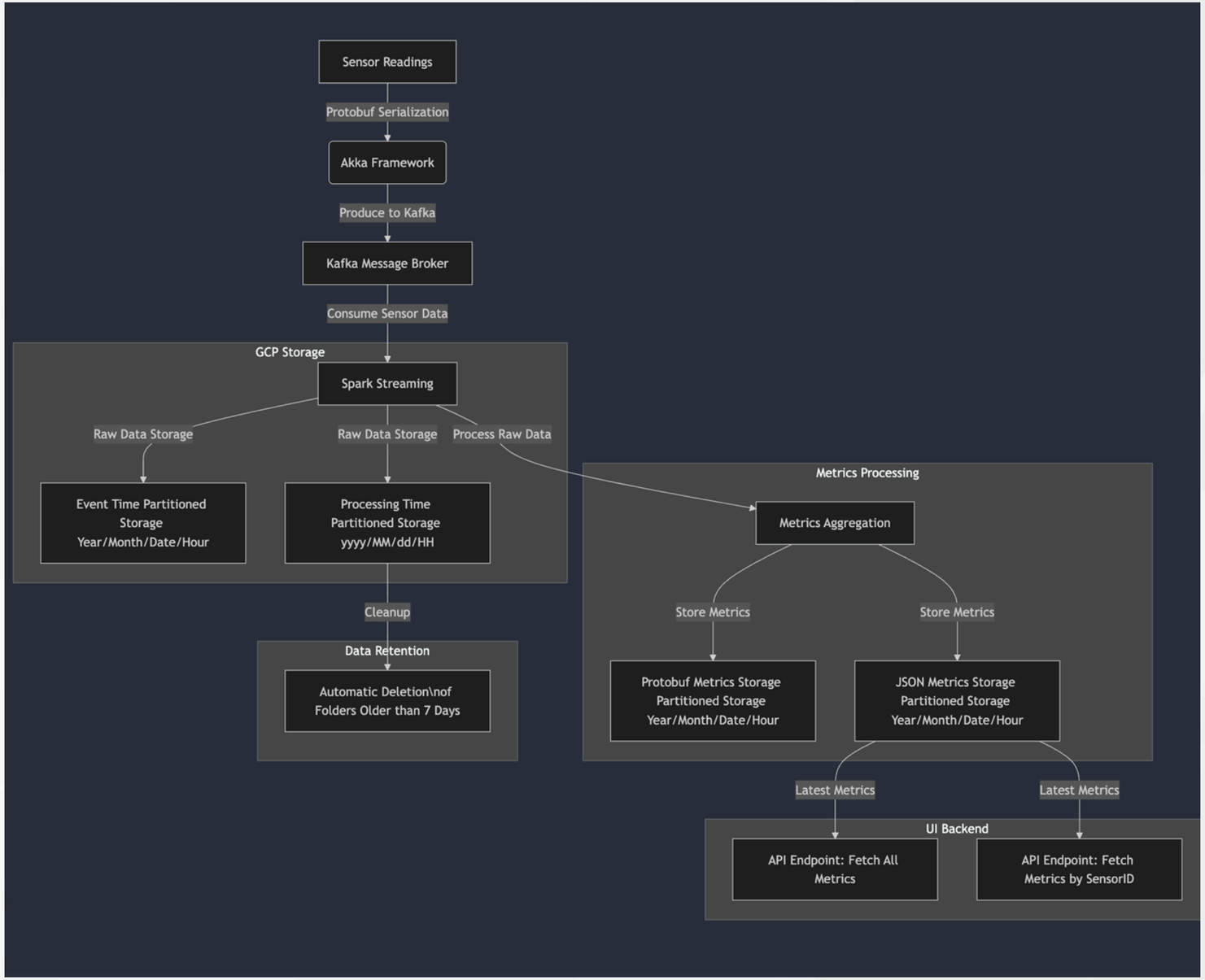
Key Features

- Real-time data generation(sensor readings) at high frequency(100ms)
- Batch processing
- Data validation, aggregation, and anomaly detection.
- APIs for serving aggregated data.
- User-friendly visualization with refresh and filter capabilities.

ARCHITECTURE OVERVIEW

- **Akka Producer**: Generates sensor data.
- **Kafka Broker**: Ensures data transmission.
- **Spark Streaming & Batch**: Processes and aggregates data.
- **Google Cloud Storage**: Stores raw and aggregated data.
- **API & Dashboard**: Serves and visualizes aggregated metrics.

- Protobuf for efficient serialization.
- JSON for dashboard compatibility.



KEY COMPONENTS

Data Generation (Akka Producer)

- Generates random sensor readings every 100ms
- Sensor Data Fields:
  - sensorId: Random integer
  - timestamp
  - temperature
  - humidity
- Publishes to Kafka topic: *sensor-readings*

```
Message 1701: SensorReading(7,1733934195879,6.331028,38.639416,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:15
Message 1702: SensorReading(6,1733934196000,85.922,36.571182,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:16
Message 1703: SensorReading(10,1733934196120,110.40265,22.28349,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:16
Message 1704: SensorReading(1,1733934196240,59.406258,91.92339,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:16
Message 1705: SensorReading(9,1733934196360,-28.67378,71.15196,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:16
Message 1706: SensorReading(2,1733934196478,17.553223,82.02908,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:16
Message 1707: SensorReading(9,1733934196599,133.90157,47.03096,UnknownFieldSet(Map())) sent at 2024-12-12 21:53:16
```

Message Brokering (Kafka)

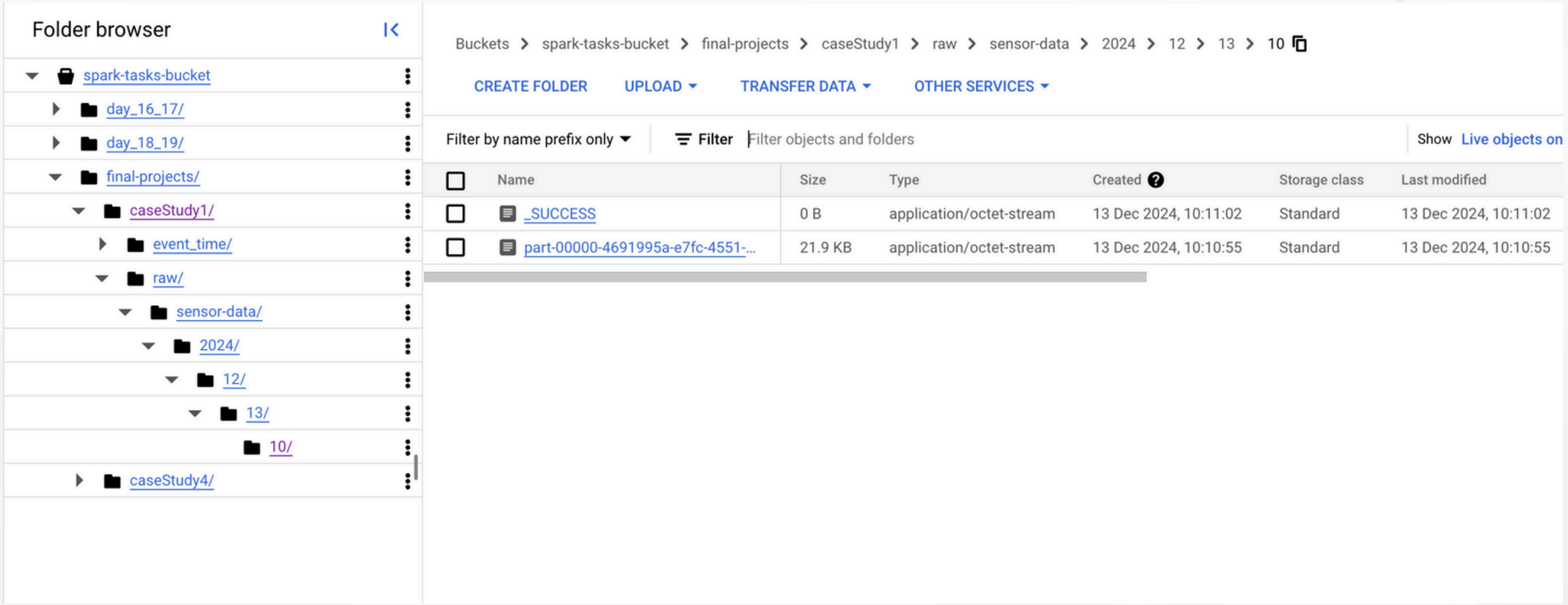
- Topic: *sensor-readings*
- Ensures high-frequency data handling
- Validated: No data loss (1707 messages successfully produced)

```
      ???2??c??%?M?B
??2m?A%??B
      ?П??2??C%?<B
Processed a total of 1707 messages
```

KEY COMPONENTS

Data Validation & Storage (Spark Streaming + GCS)

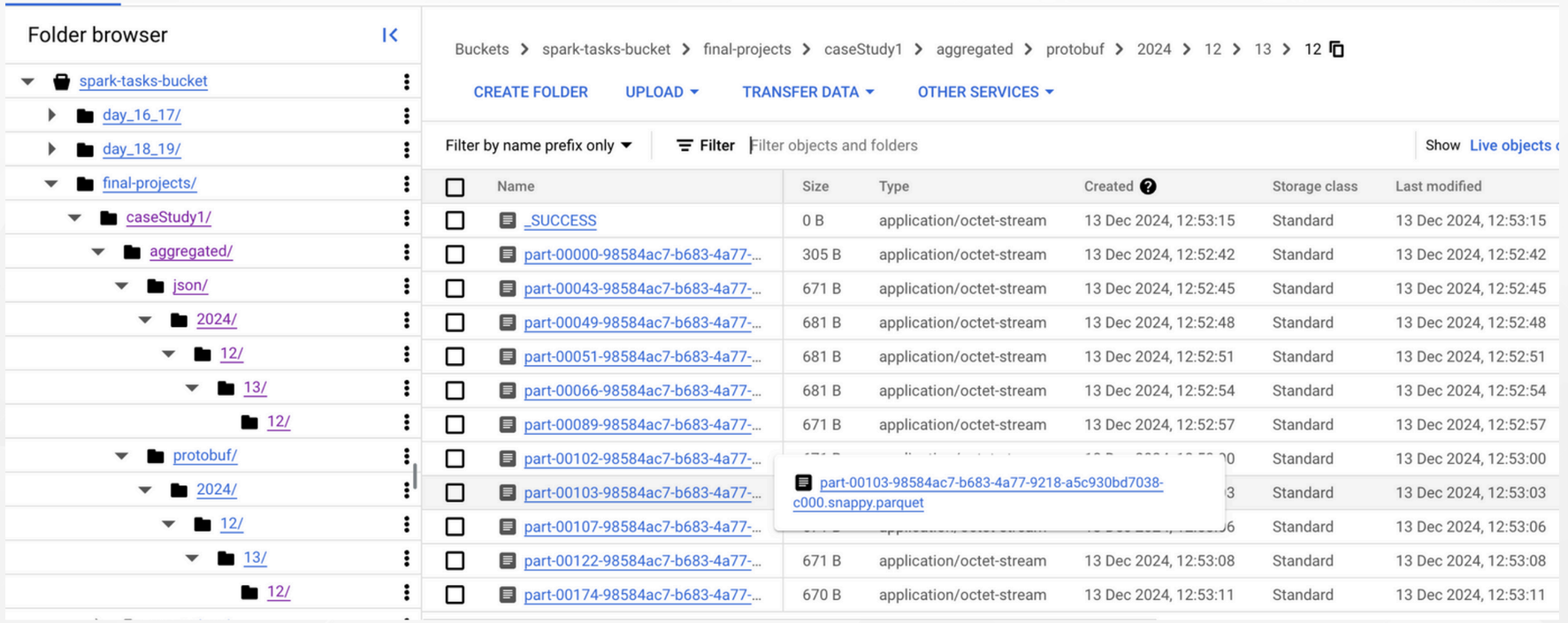
- Consumes messages from Kafka
- Validates sensor readings
- Discards out-of-range temperature/humidity
- Serialises valid readings to Protobuf
- Stores in GCS: *gs://bucket/raw/sensor-data/yyyy/MM/dd/HH/*



KEY COMPONENTS

Incremental Aggregation(Spark Batch)

- Reads raw Protobuf files
- Merges new data with existing aggregates
- Computes metrics:
  - Weighted averages
  - Min/Max values
- Stores results in:
  - Protobuf: gs://bucket/aggregated/protobuf/
  - JSON: gs://bucket/aggregated/json/



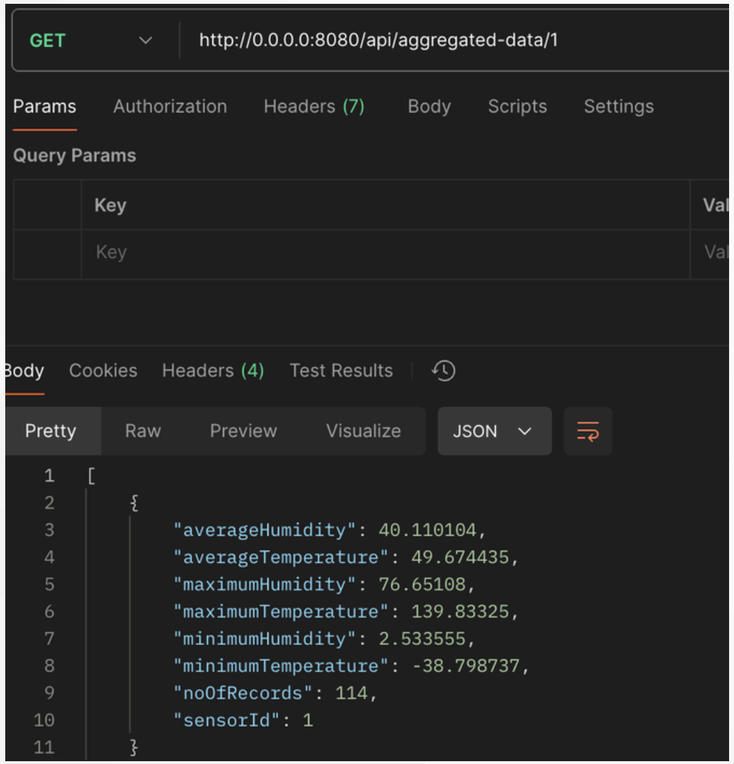
KEY COMPONENTS

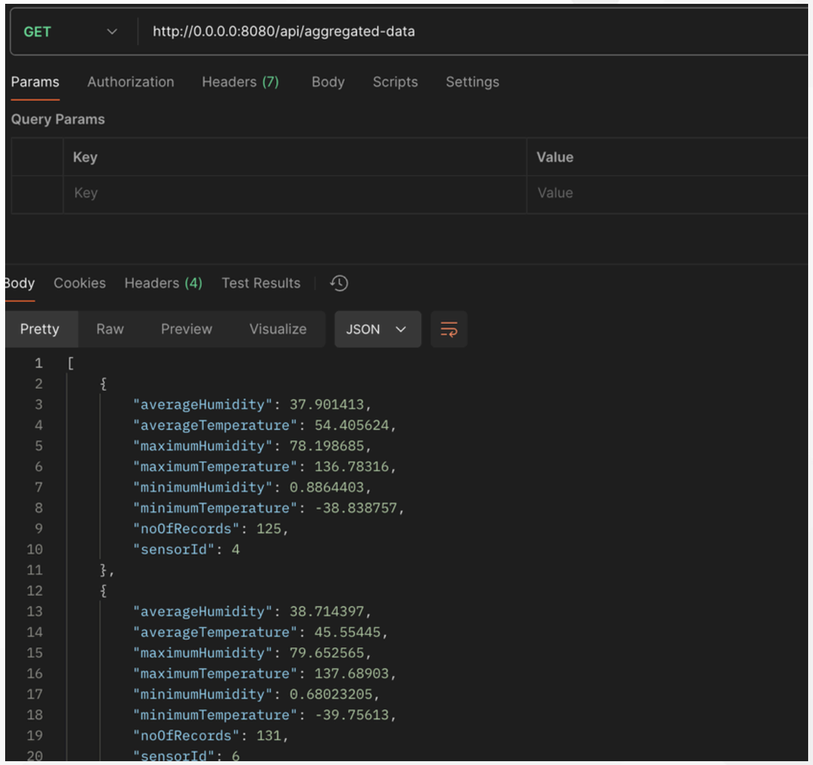
Data Retention Policy

- Deletes raw Protobuf files older than 7 days
- Retains only latest aggregated Protobuf files

API(Akka)

- GET /api/aggregated-data: Fetch latest aggregated data
- GET /api/aggregated-data/:sensorId: Fetch sensor-specific data

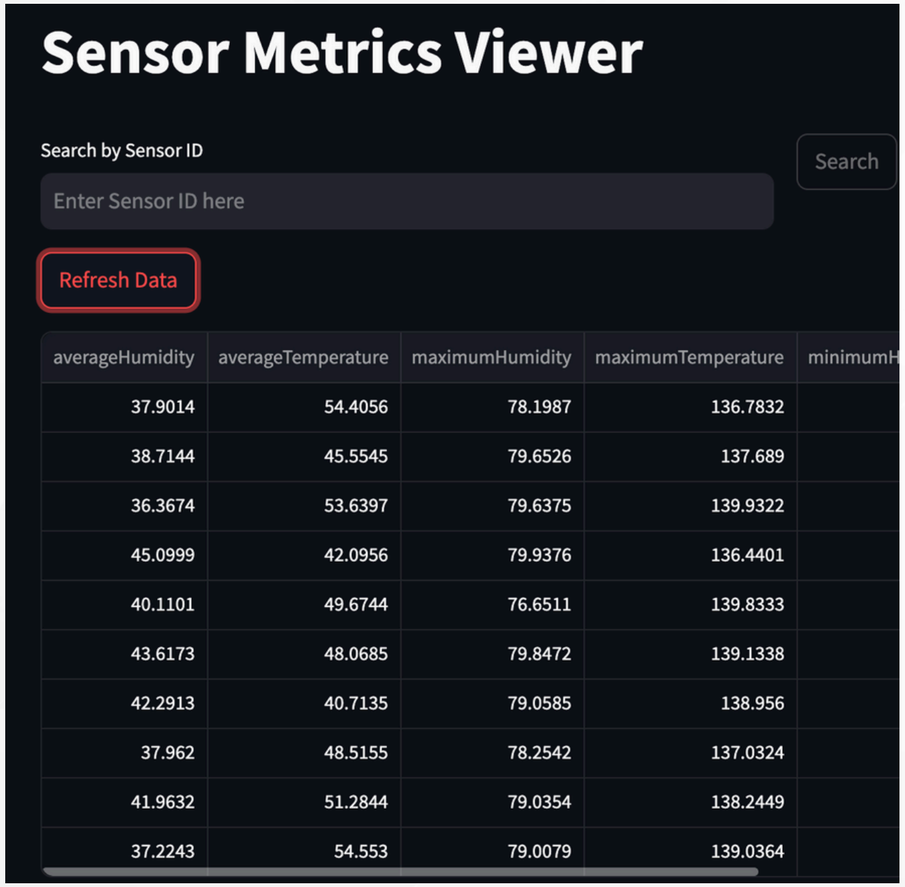


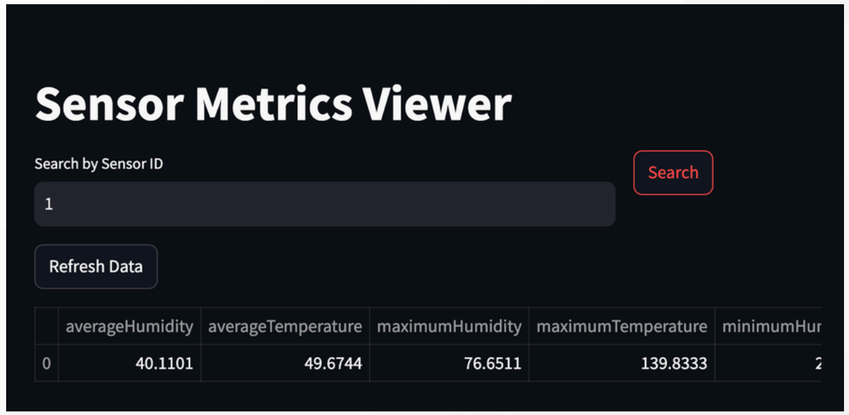


KEY COMPONENTS

Visualisation(Streamlit)

- Displays aggregated metrics in a table
- Search functionality by sensorId
- Refresh button for real-time updates





KEY OUTCOMES

- Real-time data flow achieved without loss.
- Aggregated metrics ready for visualization.
- Efficient data storage and retention implemented.

For more information or detailed implementation and observations:
Please refer to: Case Study 1 - Implementation Details

CASE STUDY

OPTIMIZED REAL-TIME DATA PROCESSING WITH CACHING, PERSISTING, AND BROADCASTING

An optimized and scalable real-time data processing pipeline for the Walmart Recruiting Sales Dataset, designed to ingest, validate, enrich, and aggregate sales data efficiently, leveraging caching, persisting, and broadcasting for performance optimization and seamless storage in GCP.

OBJECTIVES AND KEY FEATURES

Objectives

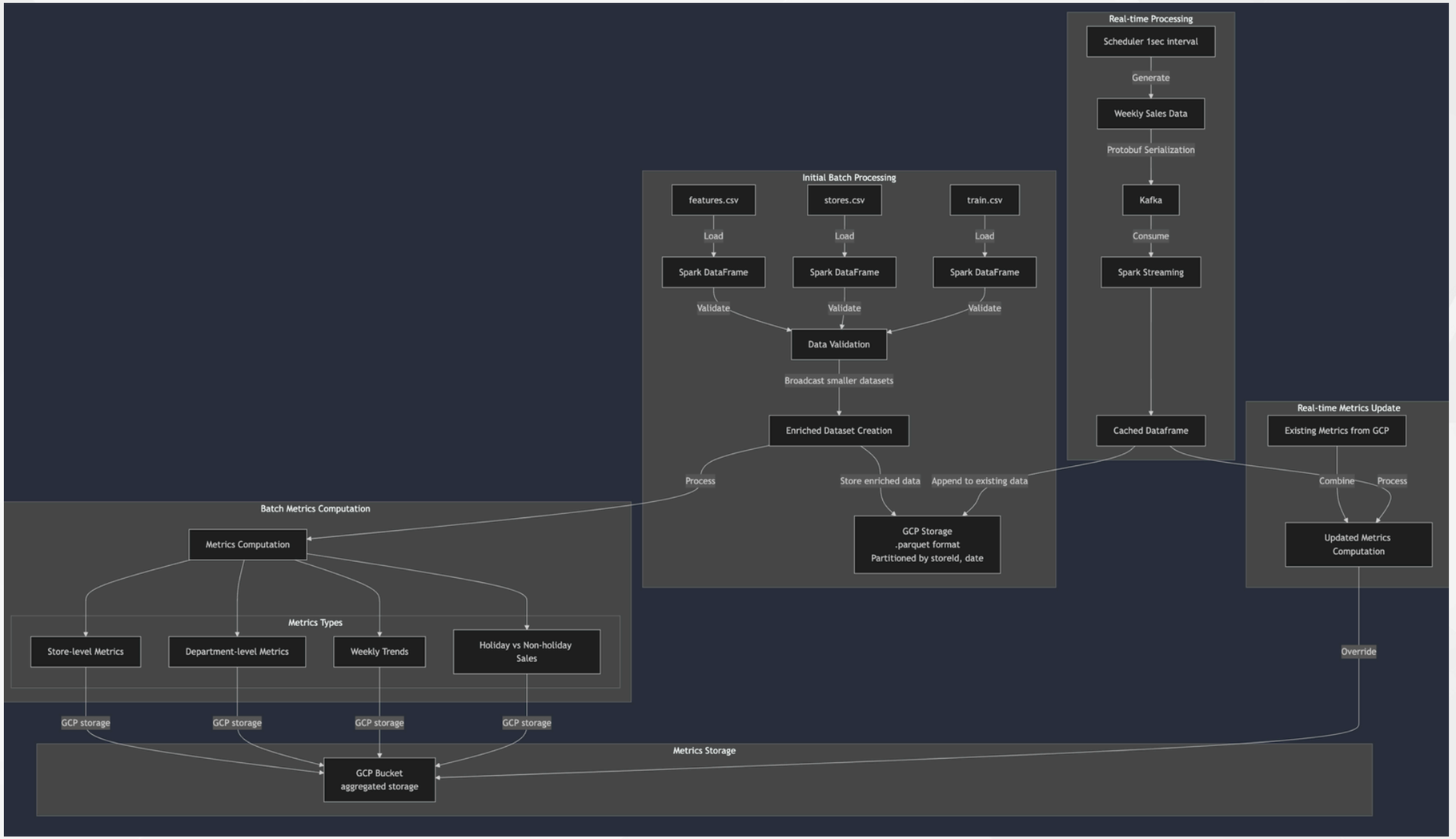
- Build a scalable pipeline for processing, validating, and enriching Walmart sales data using caching, persisting, and broadcasting to improve performance.
- Compute store-level and department-level sales metrics, including trends, top-performing stores, and holiday vs. non-holiday comparisons.
- Implement a real-time data pipeline that ingests and updates sales data using Kafka and Spark Streaming for continuous metric updates.

Key Features

- Data Validation & Enrichment
- Performance-Driven Caching:
- Sales Metrics Aggregation
- Efficient GCP Storage
- Real-Time Processing

ARCHITECTURE OVERVIEW

- Walmart Dataset
- Spark DataFrames
- Google Cloud Storage
- Metric Computation
- Real-Time Data Simulation
- Batch-Real-Time Data Integration
- Optimization Techniques



KEY COMPONENTS

Datasets

- features.csv
- stores.csv
- train.csv

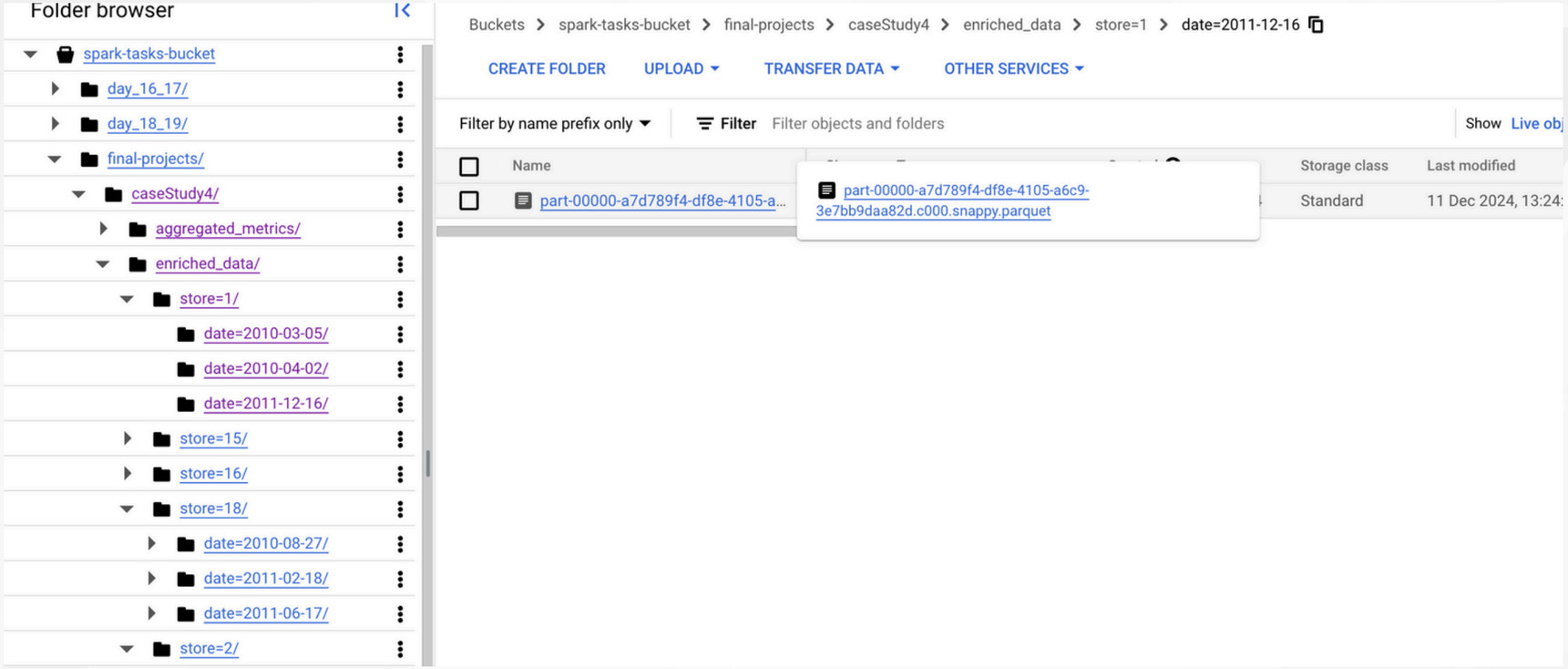
Spark DataFrames

- Data from the CSV files (features.csv, stores.csv, train.csv) are loaded into Spark DataFrames for validation, transformation, and aggregation.

Google Cloud Storage

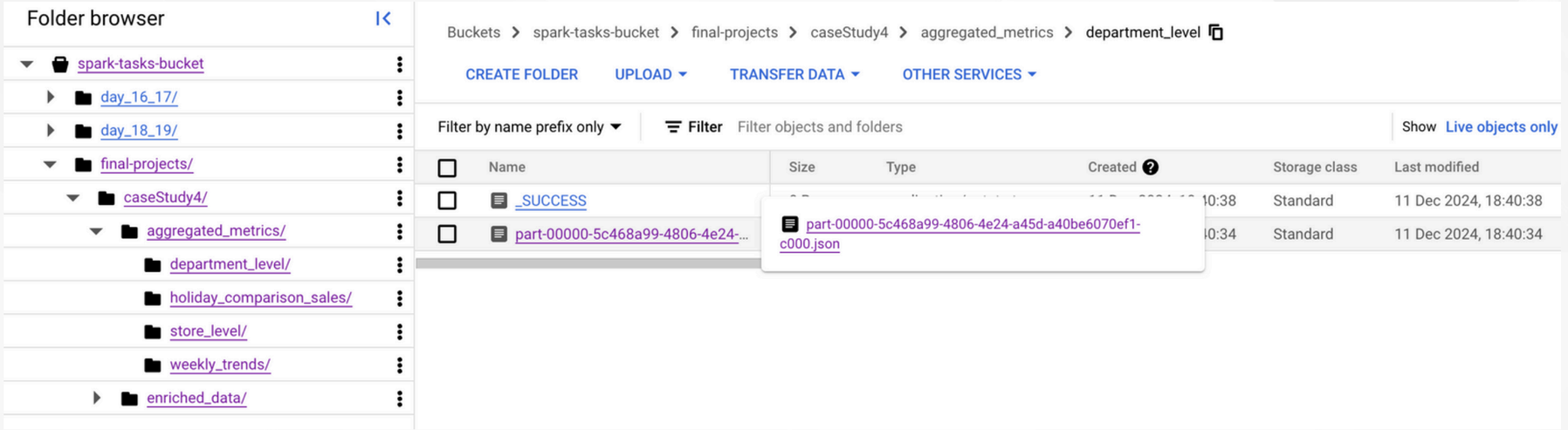
- Enriched Dataset: The cumulative dataset generated from the combination of the raw data and enrichment process is stored in GCS in Parquet format, partitioned by storeId and date.
- Aggregated Metrics: The key metrics (store-level, department-level, weekly trends, holiday vs non-holiday sales) are computed and stored in GCS in JSON format under the aggregated_metrics folder.

KEY COMPONENTS



Enriched data storage

KEY COMPONENTS



Aggregated data storage

KEY COMPONENTS

Real-Time Data Processing

- WeeklySalesData: A serialized data format (using Protobuf) for efficient transmission of weekly sales updates.
- Kafka is used to stream serialized WeeklySalesData messages in real-time.
- Spark Streaming: Consumes and processes WeeklySalesData from Kafka to update metrics.

```
Message: WeeklySalesData(5,8,2024-12-11T21:36:02.507898,1287.9885,FALSE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:02
Message: WeeklySalesData(5,6,2024-12-11T21:36:03.508156,1872.0851,TRUE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:03
Message: WeeklySalesData(2,2,2024-12-11T21:36:04.504057,1257.6008,TRUE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:04
Message: WeeklySalesData(2,8,2024-12-11T21:36:05.507686,1400.0901,TRUE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:05
Message: WeeklySalesData(4,4,2024-12-11T21:36:06.507479,1723.2825,FALSE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:06
Message: WeeklySalesData(5,5,2024-12-11T21:36:07.504767,1369.3768,FALSE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:07
Message: WeeklySalesData(1,5,2024-12-11T21:36:08.505274,1614.9503,FALSE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:08
Message: WeeklySalesData(3,9,2024-12-11T21:36:09.507516,1204.9893,TRUE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:09
Message: WeeklySalesData(1,4,2024-12-11T21:36:10.503260,1219.1056,TRUE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:10
Message: WeeklySalesData(1,6,2024-12-11T21:36:11.507544,1222.1283,FALSE,UnknownFieldSet(Map())) sent at 2024-12-11 21:36:11
```

```
2024-12-11T21:36:08.505274%1[[D*FALSE
        2024-12-11T21:36:09.507516%[[[D*TRUE
2024-12-11T21:36:10.503260%ac[D*TRUE
2024-12-11T21:36:11.507544%[D*FALSE
2024-12-11T21:36:12.507526%[a[D*TRUE
2024-12-11T21:36:13.506236%jB[D*FALSE
2024-12-11T21:36:14.504109%l*[D*FALSE
        2024-12-11T21:36:15.507548%[[D*TRUE
2024-12-11T21:36:16.504903%[J[D*FALSE
2024-12-11T21:36:17.507537%茯 D*TRUE
2024-12-11T21:36:18.507534%C[D*FALSE
2024-12-11T21:36:19.507527%P [D*FALSE
```

KEY COMPONENTS

Metrics

- Store-Level Metrics: Metrics like total and average weekly sales are computed and stored.
- Department-Level Metrics: Metrics related to individual departments, such as total sales, are computed.
- Weekly Trends: Trends in weekly sales are computed as the difference in sales from the previous week.
- Holiday vs. Non-Holiday Sales: A comparison of sales during holiday periods vs non-holiday periods is computed.

KEY OUTCOMES

- Real-time integration of sales data with continuous metric updates.
- Optimized data storage using Parquet and JSON formats in GCP.
- Efficient data processing through broadcasting, caching, and persisting.
- Efficient data storage for comprehensive insights.

For more information or detailed implementation and observations:
Please refer to: Case Study 4 - Implementation Details

THANK YOU