Project Title: End-to-End Big Data Pipeline

Course: ICS474 Big Data Analytics

Names & IDs:

Murtadha Abdulaziz ALGhadban & 202034580

Hassan Ali Abdulaal & 202042820

Section:02

Instructor: Dr. Waleed AL Gobi

Information & Computer Science Department

King Fahd University of Petroleum and Minerals

Date of submission 2025/12/19

Table of Contents

Table of Contents

Introduction

Big data analytics is important today because many systems generate continuous data from sensors and monitoring devices. Examples include environmental monitoring, industrial systems, and smart applications where measurements such as temperature, humidity, and pressure are collected continuously. This data arrives very fast, which makes traditional batch processing slow and unsuitable for real-time monitoring.

One common problem in such systems is that raw sensor data alone is difficult to manage and analyze. Without a proper streaming pipeline, it is hard to monitor sensor behavior in real time, detect abnormal readings, or store data for later analysis. This can lead to delayed responses, missing critical events, and poor understanding of system behavior over time.

This project addresses these issues by implementing an end-to-end big data analytics pipeline specifically designed to monitor streaming sensor data. The pipeline processes temperature, humidity, and pressure readings in real time, detects abnormal values, stores the processed data in a database, and visualizes the results using interactive dashboards. This enables real-time monitoring as well as historical analysis of sensor trends.

The system can be used in applications such as environmental monitoring, smart buildings, and system health monitoring, where continuous data tracking and fast detection of abnormal conditions are required. The main benefits of the project include real-time data processing, anomaly detection, persistent data storage, and clear visualization of sensor behavior.

This project builds upon an existing Kafka streaming prototype and extends it by adding Spark-based stream processing, PostgreSQL database storage, anomaly detection, and Grafana dashboards to form a complete end-to-end big data pipeline.

Pipeline Architecture

The pipeline architecture of this project is designed to monitor sensor data in real time using a simple end-to-end flow. The system is divided into clear stages, where data moves sequentially from generation to visualization for efficient processing and monitoring.

The pipeline begins with streaming sensor data representing temperature, humidity, and pressure values. This data is continuously sent to Apache Kafka, which acts as the data ingestion layer by receiving and distributing the data stream.

Apache Spark Structured Streaming consumes the data from Kafka and processes it in real time. During this stage, basic transformations and anomaly detection are applied to identify abnormal sensor readings as they occur.

The processed data is stored in a PostgreSQL database, which serves as the persistent storage layer. Both normal and anomalous readings are saved for historical analysis and querying. The database is managed using pgAdmin 4 to view tables, run queries, and verify the stored data.

Finally, Grafana connects to the PostgreSQL database and displays the sensor data through interactive dashboards. These dashboards show real-time values, trends, and anomaly indicators for easy monitoring.

Overall, the pipeline follows the flow: **Sensor Data → Kafka → Spark Streaming → PostgreSQL → Grafana**, forming a complete real-time big data analytics pipeline.

Figure 1: End-to-end pipeline showing the flow of sensor data from Kafka ingestion to Spark processing, PostgreSQL storage, and Grafana visualization.

Implementation Details & Results and Outputs

This section describes how each component of the big data pipeline was implemented and how the system was executed and verified using screenshots captured during runtime.

Kafka Setup and Data Ingestion

Apache Kafka was used as a messaging system to enable real-time data ingestion. ZooKeeper was first started to manage Kafka metadata and coordinate the Kafka services. After that, the Kafka message broker was launched to handle streaming communication between producers and consumers.

A Kafka producer was implemented in Python programming language to generate simulated streaming sensor data that includes temperature, humidity, and pressure values. The producer continuously publishes messages to a Kafka topic in real time, allowing downstream components to consume the data as it is generated.

By referring to **Figure 2**, the ZooKeeper service can be seen running in PowerShell. The log messages indicate that ZooKeeper is active and managing coordination tasks required by Kafka. This confirms that the coordination service needed for Kafka operation is successfully started.

By referring to **Figure 3**, the Kafka message broker is shown running in PowerShell. The broker logs indicate successful startup and ongoing management of consumer groups and offsets, confirming that Kafka is ready to receive and distribute streaming data.

By referring to **Figure 4**, the Kafka producer implemented in Python can be seen running in PowerShell and continuously publishing sensor data messages. The generated messages include timestamped temperature, humidity, and pressure values, demonstrating that real-time data ingestion into Kafka is functioning correctly.

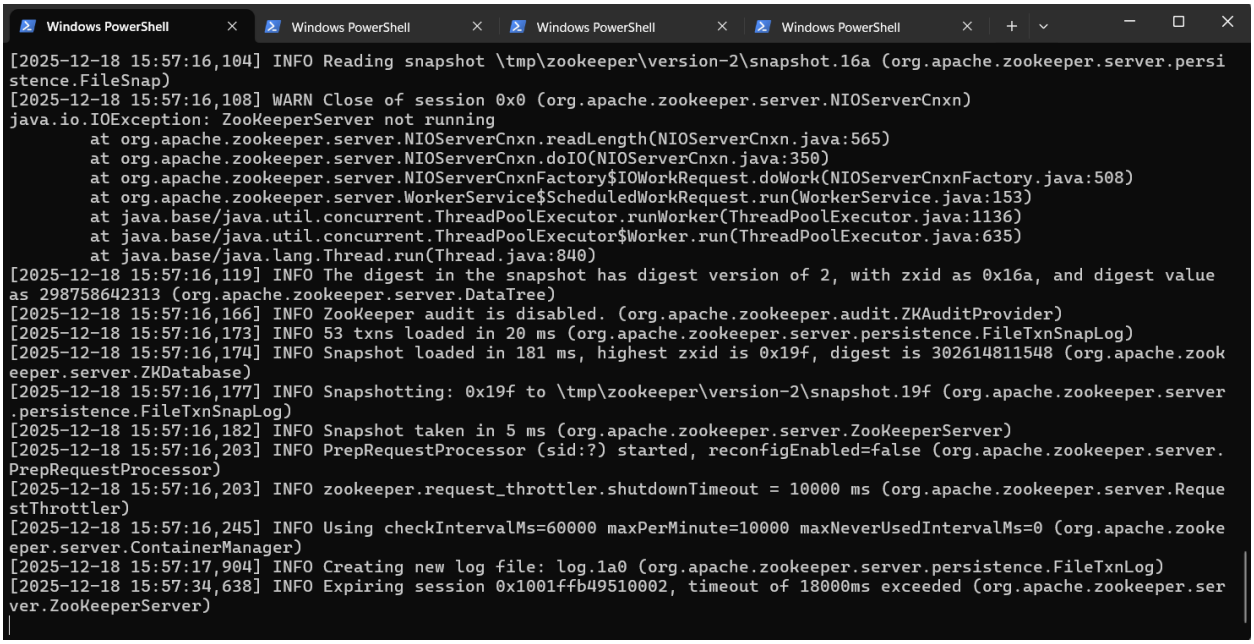


Figure 2: *ZooKeeper service running in PowerShell and coordinating Kafka services.*

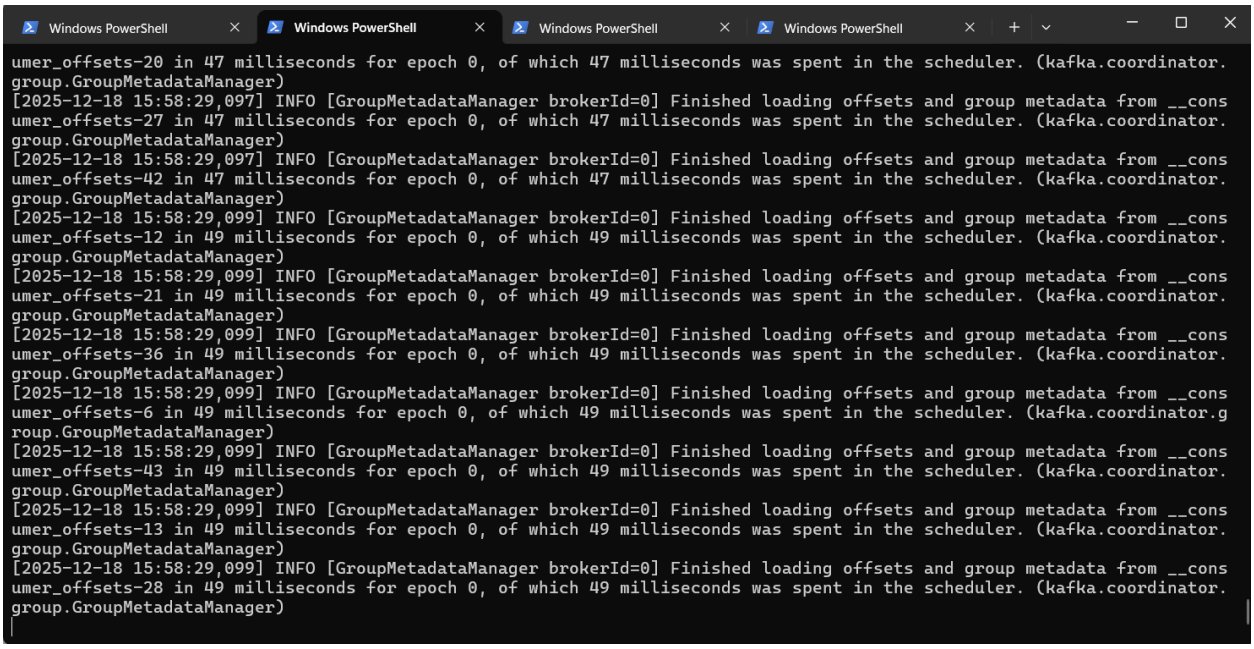


Figure 3: *Kafka message broker running in PowerShell and handling streaming data communication.*

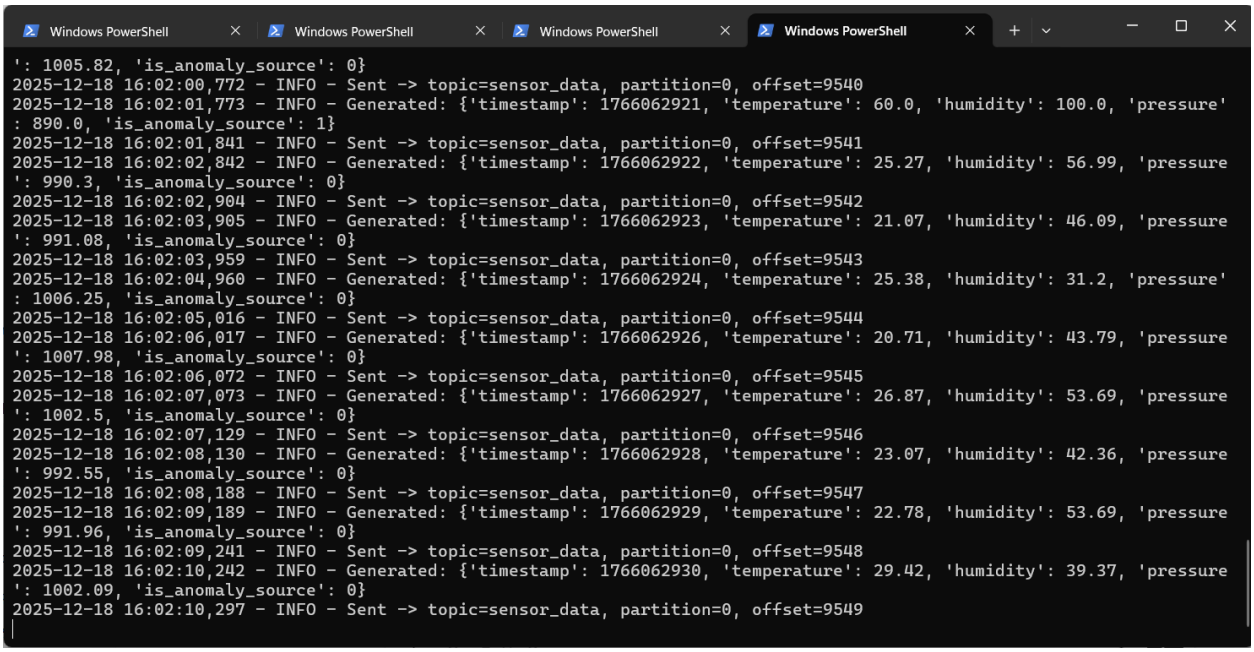


Figure 4: *Python-based Kafka producer running in PowerShell and publishing streaming sensor data to a Kafka topic.*

Spark Structured Streaming and Anomaly Detection

Apache Spark Structured Streaming was used to process the sensor data stream in real time. The streaming application was implemented using **PySpark** and executed using spark-submit to run the script pyspark_stream_to_postgres.py. This application consumes data from the Kafka topic, applies basic transformations, and implements anomaly detection logic based on predefined threshold values for the sensor readings.

Sensor readings that exceed normal operating ranges are flagged as anomalous during stream processing, before being written to the database.

By referring to **Figure 5**, the Spark Structured Streaming application is shown running using spark-submit. The log output indicates that Spark is processing incoming data in micro-batches and writing the processed results to PostgreSQL. This confirms that real-time stream processing and data storage are operating as expected.

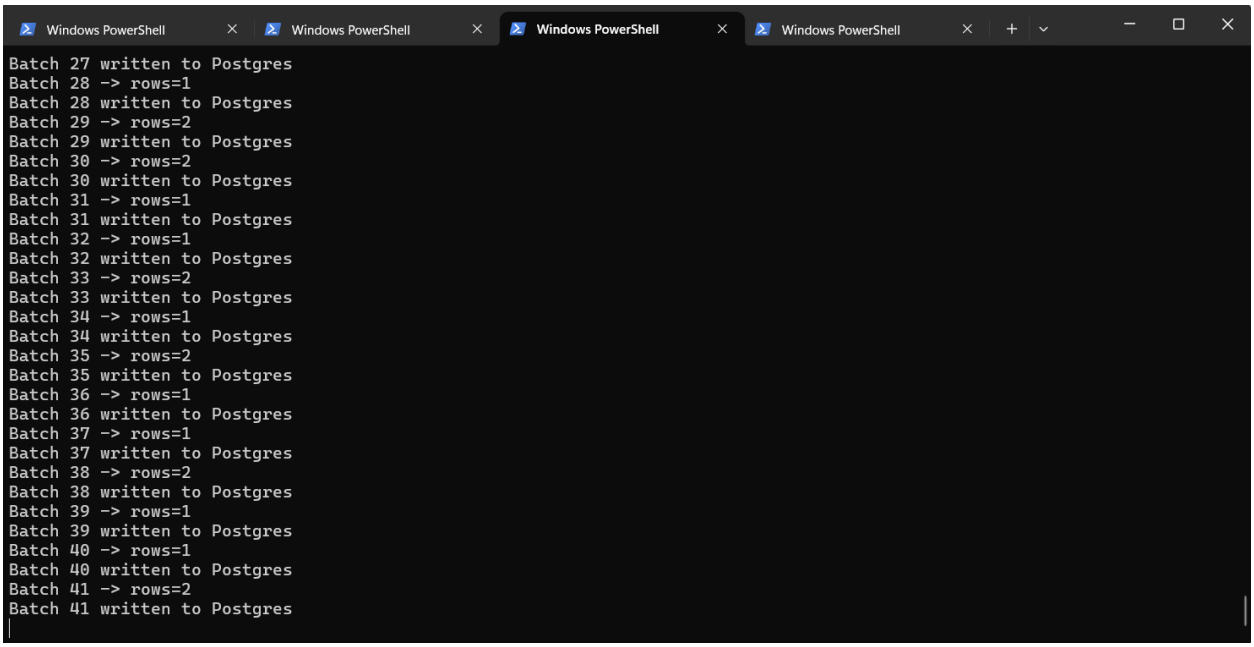


Figure 5: *Spark Structured Streaming job (pyspark_stream_to_postgres.py) running via spark-submit and writing processed data to PostgreSQL.*

PostgreSQL Storage and Database Management

After stream processing is completed, both normal and anomalous sensor readings are written to a PostgreSQL database. The database serves as the persistent storage layer of the pipeline, enabling historical analysis, querying, and integration with visualization tools.

The PostgreSQL database is managed using **pgAdmin 4**, which provides a graphical interface to create tables, execute SQL queries, and inspect stored data during execution.

By referring to **Figure 6**, the contents of the sensor_readings table are shown as retrieved using an SQL query in pgAdmin 4. The displayed records include timestamp, sensor identifier, temperature, humidity, and pressure values, along with anomaly-related fields. The presence of both normal readings and records flagged as anomalous confirms that processed data from Spark is being successfully stored in PostgreSQL and is available for further analysis.

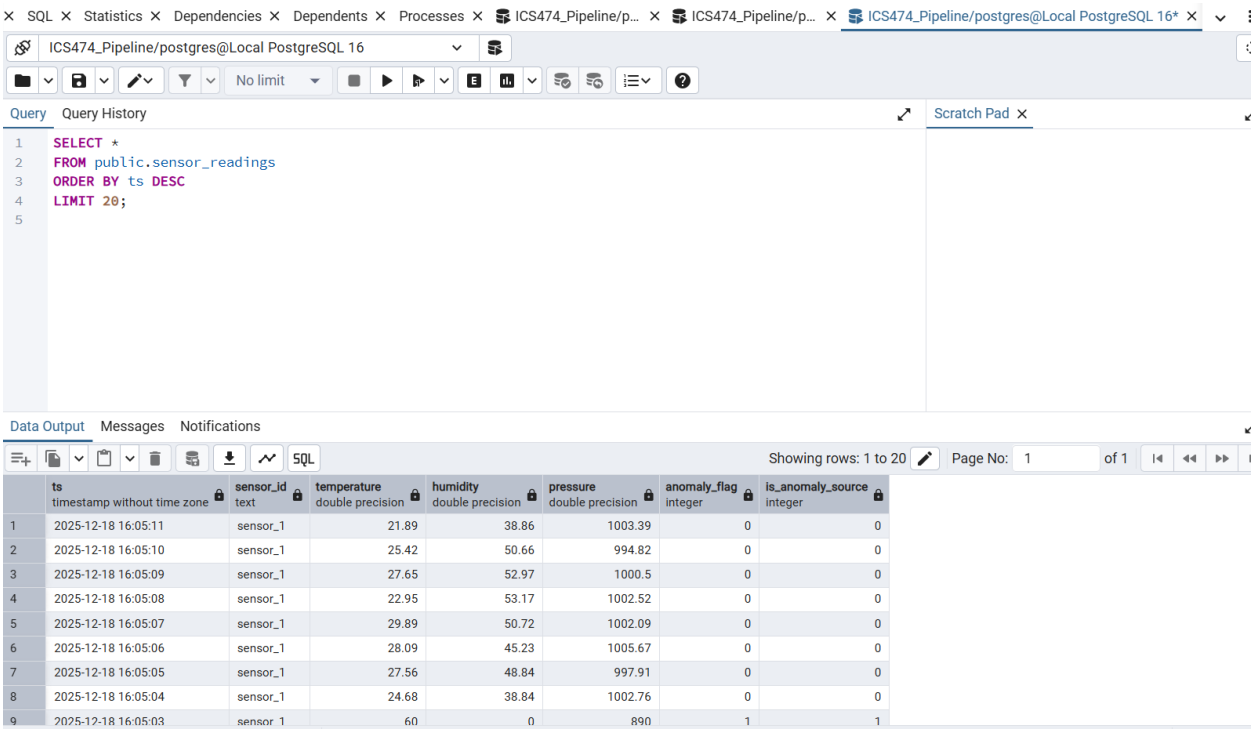


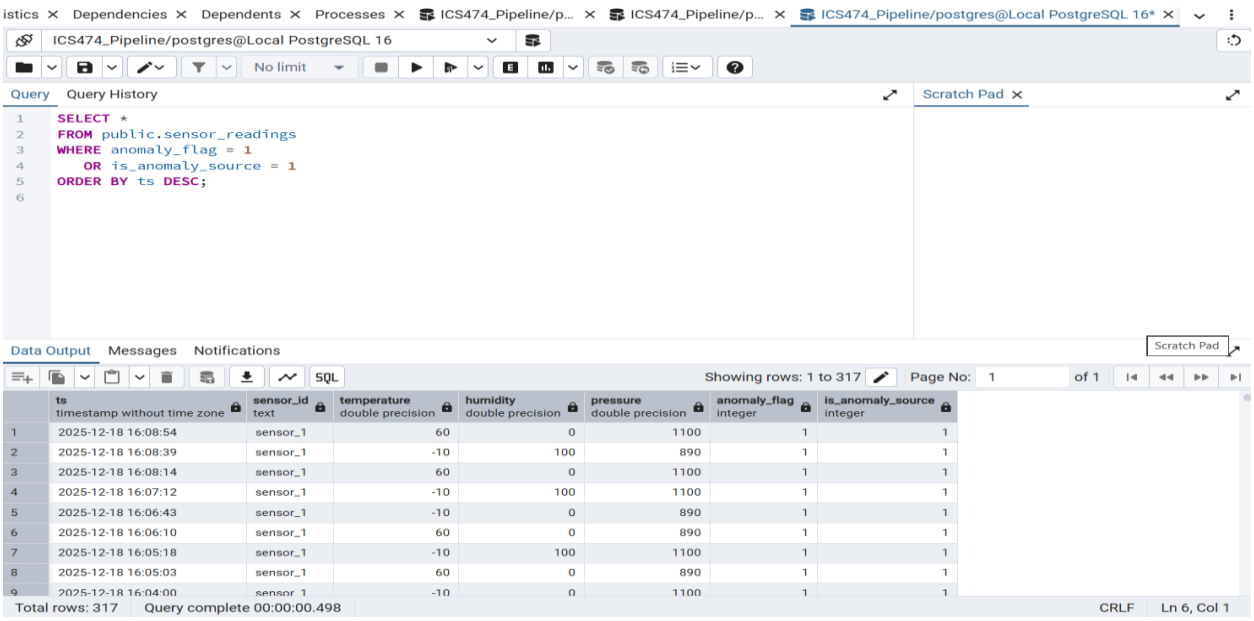
Figure 6: *PostgreSQL sensor_readings table displayed in pgAdmin 4 showing recently stored sensor data, including anomaly indicators.*

Verification of Anomaly Detection Using SQL Queries

Although the anomaly detection logic is implemented during real-time processing using Spark Structured Streaming, SQL queries executed in pgAdmin 4 are used to verify and analyze the detected anomalies stored in the PostgreSQL database. This verification step ensures that anomalous readings identified during stream processing correctly persisted.

By referring to **Figure 7**, an SQL query executed in pgAdmin 4 shows that retrieves only sensor readings flagged as anomalous. The query filters record where either the anomaly flag or is_anomaly_source field is set to 1 and orders the results by timestamp. The displayed rows clearly show extreme sensor values, such as unusually high or low temperature, humidity, and pressure readings, which confirms that the anomaly detection logic applied in Spark is correctly reflected in the stored database records.

The results demonstrate that anomalous sensor data is successfully identified during stream processing, stored in PostgreSQL, and made available for further analysis and visualization.



	ts timestamp without time zone	sensor_id text	temperature double precision	humidity double precision	pressure double precision	anomaly_flag integer	is_anomaly_source integer
1	2025-12-18 16:08:54	sensor_1	60	0	1100	1	1
2	2025-12-18 16:08:39	sensor_1	-10	100	890	1	1
3	2025-12-18 16:08:14	sensor_1	60	0	1100	1	1
4	2025-12-18 16:07:12	sensor_1	-10	100	1100	1	1
5	2025-12-18 16:06:43	sensor_1	-10	0	890	1	1
6	2025-12-18 16:06:10	sensor_1	60	0	890	1	1
7	2025-12-18 16:05:18	sensor_1	-10	100	1100	1	1
8	2025-12-18 16:05:03	sensor_1	60	0	890	1	1
9	2025-12-18 16:04:00	sensor_1	-10	0	1100	1	1

Figure 7: *SQL query executed in pgAdmin 4 displaying sensor readings flagged as anomalous based on Spark streaming anomaly detection logic.*

Grafana Visualization

Grafana is used as the visualization layer of the pipeline and is accessed locally through http://localhost:3000. Grafana connects to the PostgreSQL database and retrieves the stored sensor data to present it in interactive dashboards. Separate panels are used to visualize temperature, humidity, and pressure readings in real time, with anomalous values clearly distinguished from normal readings.

By referring to **Figure 8**, the live humidity monitoring panel is illustrated. Normal humidity readings are shown as a continuous line, while anomalous readings are highlighted separately. This visualization makes sudden or abnormal humidity changes easy to identify.

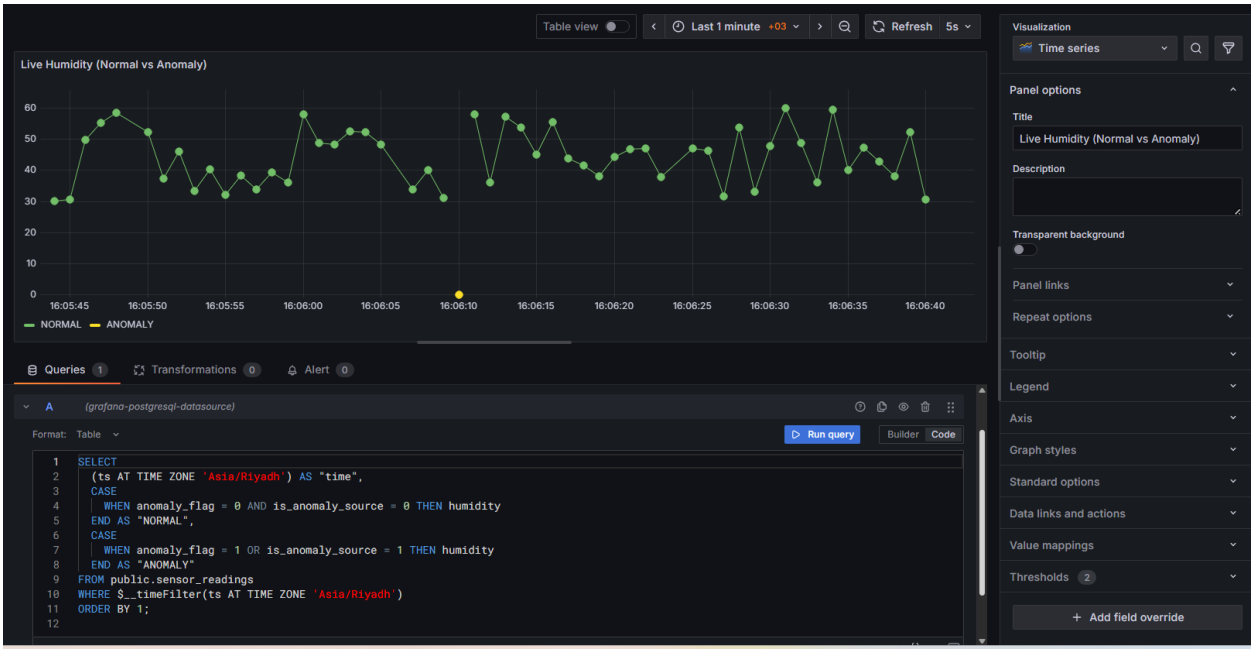


Figure 8: *Grafana panel showing live humidity readings with normal and anomalous values highlighted.*

By referring to **Figure 9**, the live pressure monitoring panel is shown. The graph displays stable pressure behavior under normal conditions, with anomalous pressure values appearing as distinct points, indicating deviations detected by the system.

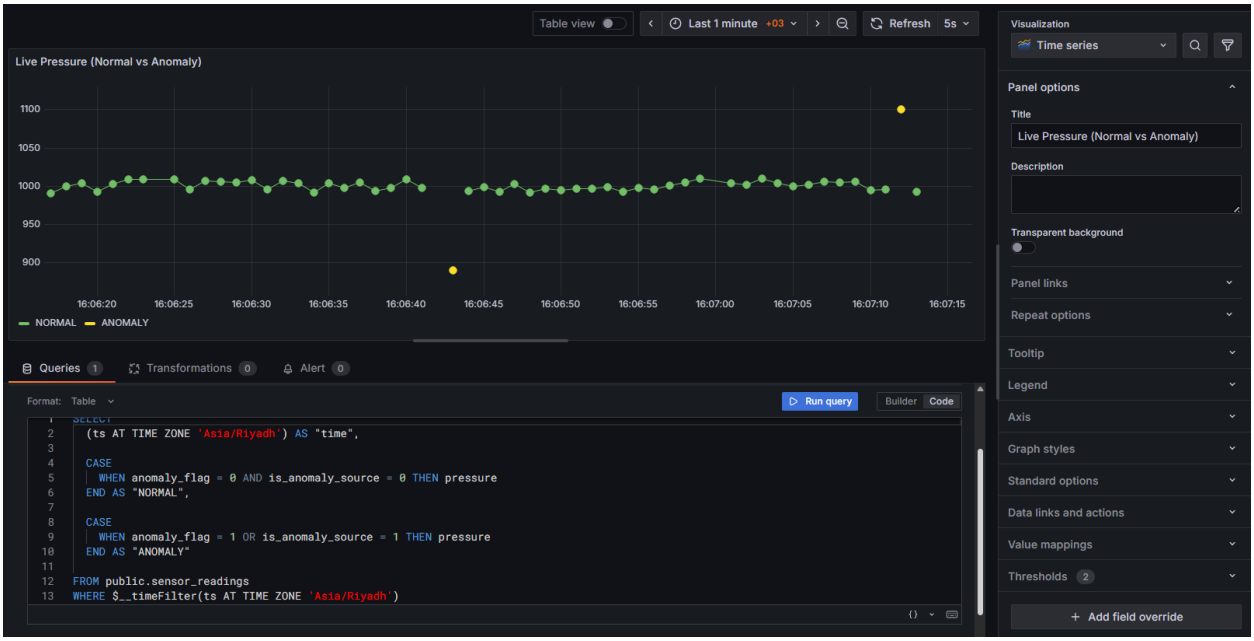


Figure 9: *Grafana panel showing live pressure readings with normal and anomalous values highlighted.*

By referring to **Figure 10**, the live temperature monitoring panel is illustrated. The visualization shows temperature trends over time and highlights anomalous temperature readings, allowing immediate identification of unexpected temperature changes.

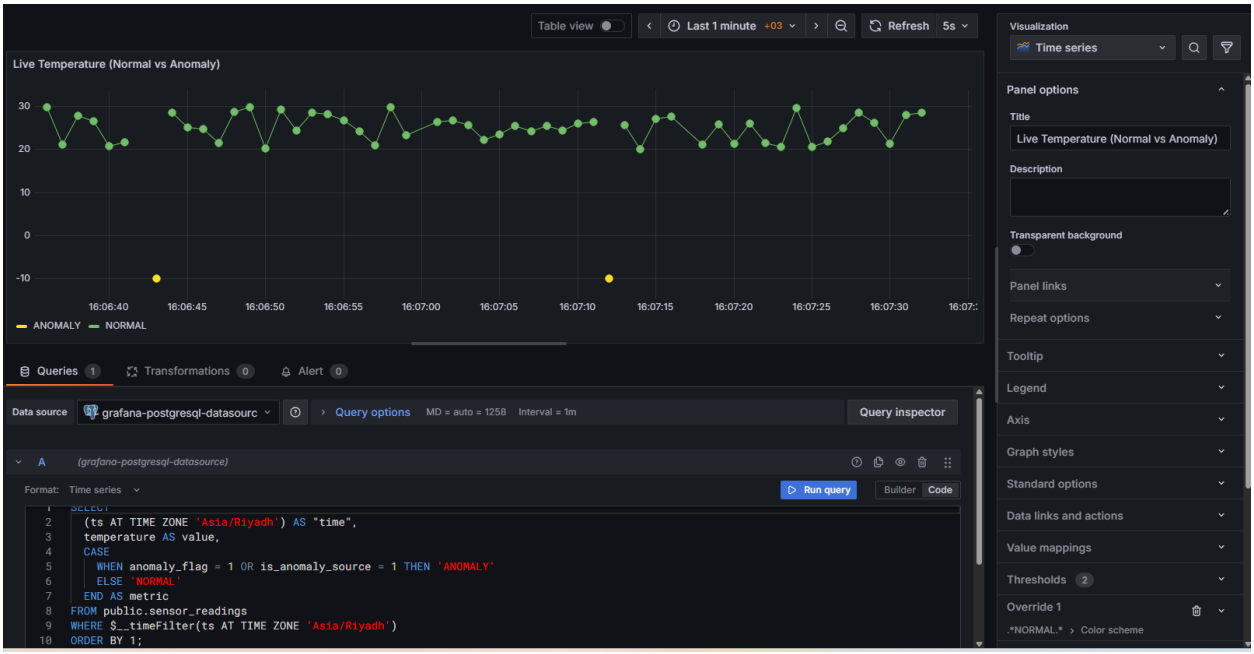


Figure 10: *Grafana panel showing live temperature readings with normal and anomalous values highlighted.*

By referring to **Figure 11**, a statistical panel showing one-minute summary metrics for normal sensor readings is displayed. This panel provides a baseline view of average, minimum, and maximum values under normal operating conditions.

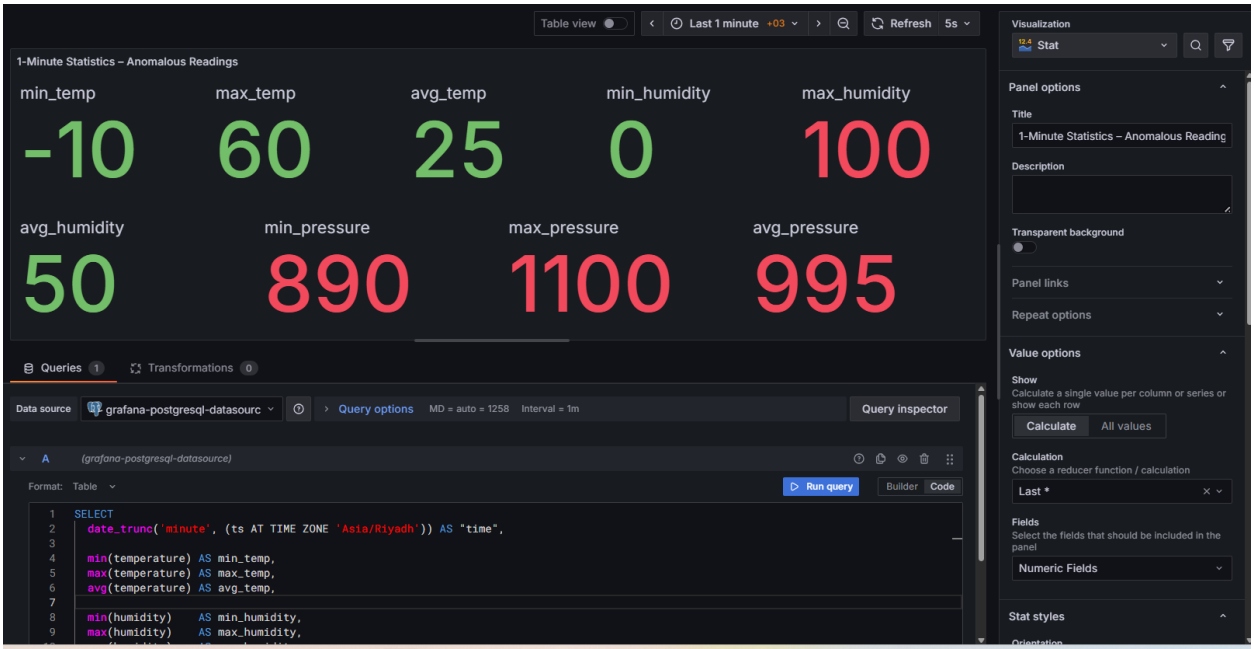


Figure 11: *Grafana panel showing short-interval summary metrics for normal sensor readings.*

By referring to **Figure 12**, a similar statistical panel for anomalous sensor readings is shown. The extreme minimum and maximum values displayed in this panel highlight the effect of detected anomalies on the overall sensor behavior.

Together, these Grafana panels confirm that sensor data processed by Spark and stored in PostgreSQL is successfully retrieved, classified, and visualized, enabling effective real-time monitoring and analysis.

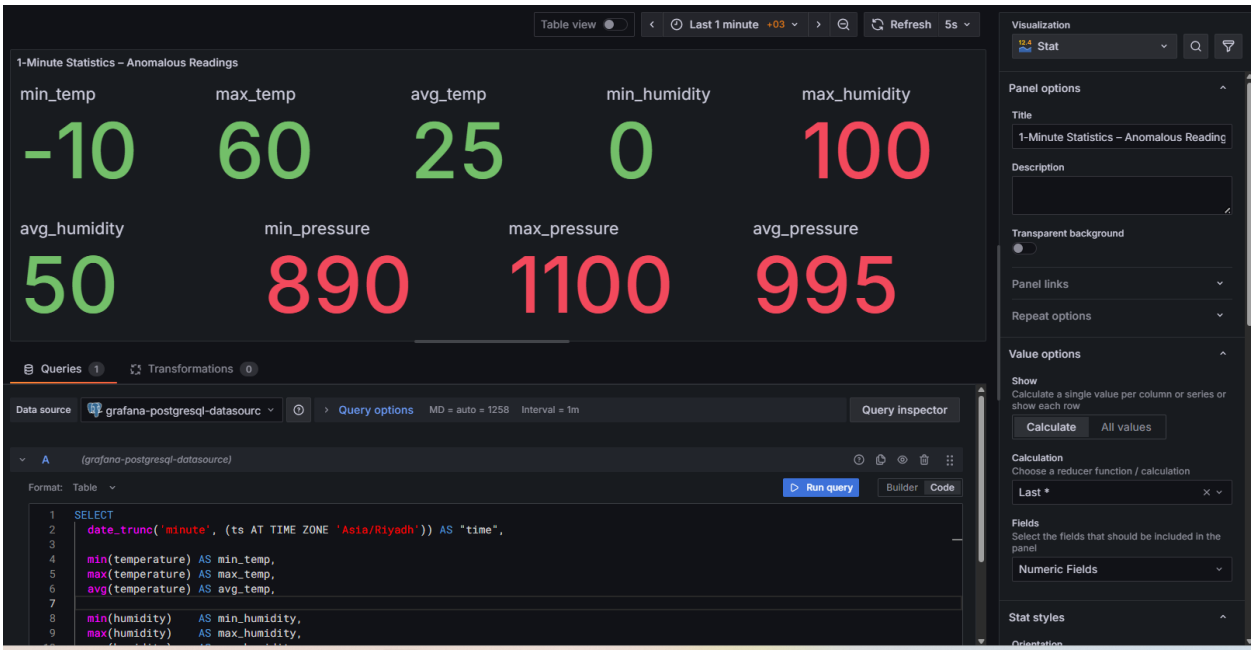


Figure 12: Grafana panel showing short-interval summary metrics for anomalous sensor readings.

Conclusion and Future Improvements

Conclusion

This project successfully implemented a complete end-to-end big data analytics pipeline for real-time sensor data monitoring. The system integrates Apache Kafka for data ingestion, Spark Structured Streaming for real-time processing and anomaly detection, PostgreSQL for persistent storage, and Grafana for visualization.

The pipeline was executed locally using PowerShell, where ZooKeeper, the Kafka broker, the Python-based Kafka producer, and the Spark streaming application were run together. Sensor data containing temperature, humidity, and pressure values was continuously streamed, processed in real time, stored in the database, and visualized through interactive dashboards.

The results demonstrate that the system can reliably detect anomalous sensor readings during stream processing and make them available for querying and visualization. The use of pgAdmin 4 and Grafana further confirms that the processed data is correctly stored and accessible for analysis. Overall, the project meets the objectives of building a functional and practical real-time big data pipeline.

Future Improvements

1- Implement more advanced anomaly detection techniques, such as statistical methods or machine learning models, instead of fixed threshold-based detection to improve accuracy.
2- Extend the pipeline to support multiple sensors and higher data volumes in order to better simulate real-world monitoring scenarios.
3- Deploy the system on a distributed Apache Kafka and Spark cluster rather than a local environment to improve scalability, reliability, and fault tolerance.
4- Add alerting mechanisms in Grafana to notify users automatically when anomalous sensor readings are detected.
5- Incorporate batch processing and long-term data analytics to enable historical analysis and trend-based insights

References

[1] Apache Kafka Documentation. *Apache Kafka*.
Available: https://kafka.apache.org/41/getting-started/

[2] Apache Spark Documentation. *Spark Structured Streaming*.
Available: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

[3] PostgreSQL Documentation. *PostgreSQL 16*.
Available: https://www.postgresql.org/docs/

[4] pgAdmin Documentation. *pgAdmin 4*.
Available: https://www.pgadmin.org/docs/

[5] Grafana Documentation. *Grafana Visualization Platform*.
Available: https://grafana.com/docs/

[6] Apostu, A. *Kafka Streaming Dashboard Prototype*. GitHub repository.
Available: https://github.com/Apostu0Alexandru/kafka-streaming-dashboard