Real-Time IoT Data Pipeline

End-to-End Streaming Architecture with Kafka, Airflow, and MySQL

Team Information

Team Member: Mohamed Bassam Abdelaziz
Team Member: Salma Ahmed Abdelsalam
Team Member: Mohamed Abdullah Ahmed
Team Member: Mohamed Tamer
Team Member: Abbdullah khedr

Supervisor: Ahmed Azab

Institution: DEPI
Track: Data Engineering

November 16, 2025

Abstract

This report details the design, implementation, and evaluation of an end-to-end, real-time data pipeline for Internet of Things (IoT) applications. The system leverages a modern data stack, including Apache Kafka for high-throughput data ingestion, Apache Airflow for robust ETL orchestration, and MySQL for persistent, structured storage. The architecture simulates temperature and humidity sensor data, processes it in mini-batches, and presents live insights via a Streamlit dashboard. The entire environment is containerized using Docker Compose, ensuring reproducibility and scalability. This project successfully demonstrates a scalable, production-ready solution for handling continuous data streams, providing a foundation for advanced analytics and real-time anomaly detection in smart environments.

Contents

1 Introduction

Real-time data processing has become a critical requirement in modern Internet of Things (IoT) ecosystems. As sensor networks grow in scale, organizations need pipelines capable of ingesting continuous data streams, orchestrating transformations, storing processed outputs, and presenting insights instantly.

This project implements an **end-to-end Real-Time IoT Data Pipeline** integrating Apache Kafka, Apache Airflow, MySQL, and Streamlit to create a scalable, production-oriented architecture for handling live temperature and humidity sensor data.

The system simulates IoT readings, streams them through Kafka, processes them using Airflow (batch ETL), stores them in MySQL, and visualizes them on a live dashboard. The entire stack runs on Docker and is monitored with Kafka UI and Airflow Web UI.

2 Project Objectives

The project aims to build a fully operational real-time data pipeline with the following objectives:

1. **Simulate real IoT sensor behavior** (temperature & humidity)
2. **Ingest data in real time** using Apache Kafka
3. **Apply both batch and streaming logic** to process incoming data
4. **Store cleaned and structured readings** in a relational database (MySQL)
5. **Visualize live updates through a real-time dashboard** (Streamlit)
6. **Orchestrate workflows using Airflow DAGs**
7. **Containerize the full environment** using Docker Compose
8. **Provide monitoring** through Kafka UI and Airflow Web UI

This architecture mirrors real production pipelines used in smart agriculture, industrial automation, and environmental monitoring.

3 System Architecture

The pipeline consists of **five main layers**:

3.1 Sensor Layer (Data Generation)

- Python script that simulates IoT device readings
- Generates timestamped temperature and humidity data every 5 seconds
- Adds simple alert logic (temperature $> 30°C$)

3.2 Messaging Layer (Kafka Broker)

- Kafka acts as the real-time ingestion system
- Sensor data is sent to a Kafka topic `iot_sensor_data`
- Kafka UI monitors messages and partitions

3.3 Orchestration Layer (Apache Airflow)

- A DAG processes data in mini-batches (every 10 minutes)
- The ETL script extracts messages from Kafka, transforms them, and loads them into MySQL

3.4 Storage Layer (MySQL Database)

- Stores historical readings for analysis and reporting
- Table: `sensor_readings` (timestamp, temperature, humidity)

3.5 Presentation Layer (Streamlit Dashboard)

- Live dashboard that refreshes every few seconds
- Shows KPIs, line charts, and real-time alerts

Architecture Diagram

[salma will make it]

4 Project Milestones

4.1 Milestone 1 — Data Simulation & Kafka Ingestion

Goal: Generate realistic IoT sensor readings and stream them to Kafka.
Files: `dags/Datagenerator.py`
Key Details:

- Generates two metrics:
  - **Temperature:** 20–40°C
  - **Humidity:** 40–80%
- Sends JSON messages every 5 seconds
- Publishes to Kafka topic: `iot_sensor_data`
- Uses KafkaProducer with `localhost:9092`

Alert Logic: If temperature $>$ 30°C $\rightarrow$ prints real-time alert to terminal.
Verification: Kafka UI used to view incoming messages in the topic.

4.2 Milestone 2 — Batch ETL Pipeline Using Airflow

Goal: Build a production-style ETL workflow.
Files: `dags/batch_etl_pipeline.py`
Process performed by ETL:

- Consumes data from Kafka (KafkaConsumer)
- Parses JSON messages
- Inserts structured records into MySQL table
- Commit happens after each batch of messages

DAG Scheduling: Runs every 10 minutes.
Airflow Interface: `localhost:8080` to monitor DAG health, logs, and run history.

4.3 Milestone 3 — Real-Time Streaming & Alerts

Goal: Provide instant anomaly detection.
Improvements:

- Embedded alert logic in data generator
- Real-time threshold checking (Temp $>$ 30°C)
- Alerts appear on terminal, in Kafka UI, and in Streamlit dashboard

Streaming Logic: Keeps monitoring live data while ETL stores history.

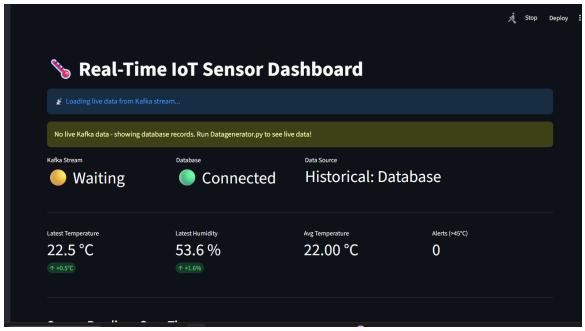
4.4 Milestone 4 — Dashboard and Reporting

Goal: Visualize metrics and insights in real time.
Files: `dashboard/app.py`
Dashboard Features:

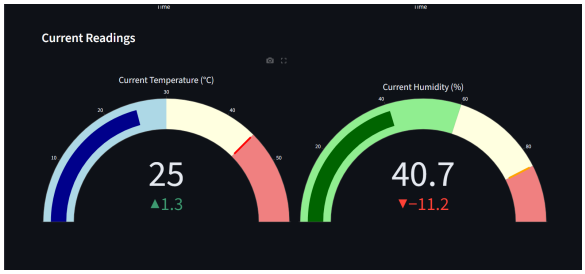
- Updates automatically every 5 seconds
- KPIs: Average Temperature, Average Humidity, Active alert count
- Line charts for real-time readings
- Temperature gauge visualization
- Historical values pulled from MySQL

Access: `localhost:8501`

Dashboard Visuals

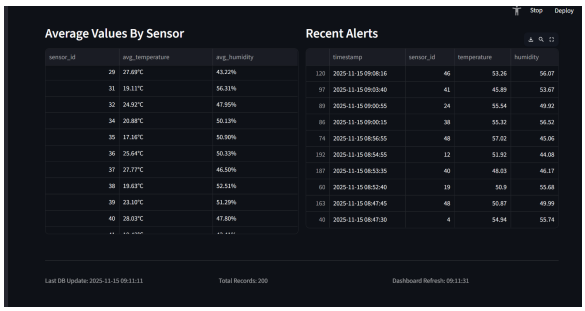


(a) Dashboard status panel showing connection and KPI summary.



(b) Real-time temperature and humidity gauges.

(c) Main dashboard with KPIs and sensor readings over time.



(d) Historical data view with average values by sensor and recent alerts.

Figure 1: Streamlit Dashboard Components

5 Repository Structure

Folder	Description
`dags/`	Contains data generator and ETL DAG scripts
`dashboard/`	Streamlit real-time dashboard
`docker-compose.yml`	Spins up Kafka, ZooKeeper, MySQL, Airflow
`screenshots/`	Monitoring dashboards and pipeline results
`docs/`	Final PDF report + presentation
`config/`	Kafka & Airflow configurations

Every component is modular and designed for reuse in larger IoT systems.

6 Detailed Component Explanation

6.1 Kafka Topic

Topic Name: `iot_sensor_data`
Partitions: 3
Replication Factor: 1

Reason for 3 partitions:

- Increases throughput
- Allows future scaling
- Distributes load across brokers

Kafka ensures real-time delivery even under heavy traffic.

6.2 Airflow DAG

The DAG contains a single PythonOperator that:

1. Connects to KafkaConsumer
2. Reads messages continuously
3. Converts JSON → Python dict
4. Inserts into MySQL table
5. Logs data processing steps

It acts as a **mini-batch ETL**, not pure streaming, which is ideal for analytics.

6.3 MySQL Schema

Table: `sensor_readings`

Column	Type
timestamp	DATETIME
temperature	FLOAT
humidity	FLOAT

Data stored here is used for dashboards and historical analytics.

6.4 Streamlit Dashboard

The dashboard queries MySQL, then renders:

- Real-time KPIs
- Line plots for both metrics

- Current alert status
- Temperature gauge
- Historical trends

Streamlit re-renders automatically every refresh cycle.

7 How to Run the Full Pipeline

Listing 1: Pipeline Execution Commands

```
# Start all services
docker-compose up -d

# Create Kafka topic
docker-compose exec kafka \
kafka-topics --create --topic iot_sensor_data \
--bootstrap-server localhost:29092 \
--partitions 3 --replication-factor 1

# Run simulator
python dags/Datagenerator.py

# Run Airflow DAG (via UI at localhost:8080)

# Run Streamlit app
cd dashboard
streamlit run app.py
```

8 Results

Once running:

- Kafka continuously receives sensor data
- Airflow processes batches successfully
- MySQL stores all readings
- Streamlit updates every few seconds with fresh metrics
- Alerts appear when temperature exceeds threshold
- Kafka UI confirms message flow
- Airflow UI logs DAG health

The system behaves like a production IoT monitoring setup.

9 Challenges & Solutions

Challenge	Solution
Kafka connection stability	Adjusted bootstrap server to `localhost:29092` in containers
Airflow to Kafka compatibility	Ensured Python environment inside Airflow had `kafka-python`
Streamlit refresh rate	Set refresh interval carefully to avoid dashboard flickering

10 Future Improvements

1. Add **Spark Streaming** for advanced analytics
2. Introduce **machine learning models** for anomaly detection
3. Implement **MQTT protocol** for more realistic IoT communication
4. Extend dashboard with geographical maps
5. Add authentication and RBAC for dashboard and Airflow

11 Conclusion

This project delivers a complete, scalable, and production-like Real-Time IoT Data Pipeline. It handles real-time ingestion, batch ETL, persistent storage, and live visualization while maintaining reliability and modularity.

The architecture is highly extensible, suitable for smart agriculture, industrial IoT, environmental monitoring, and any situation requiring continuous sensor tracking. The modular design allows for easy adaptation to different IoT use cases and scaling requirements.

12 References

References

[1] Apache Software Foundation. “Apache airflow documentation,” Accessed: Nov. 15, 2025. [Online]. Available: https://airflow.apache.org/docs/.

[2] Docker Inc. “Docker compose documentation,” Accessed: Nov. 15, 2025. [Online]. Available: https://docs.docker.com/compose/.

[3] N. Narkhede, G. Shapira, and T. Palino, *Kafka: The Definitive Guide*. Sebastopol, CA: O’Reilly Media, 2017.

[4] Oracle Corporation. “Mysql 8.0 reference manual,” Accessed: Nov. 15, 2025. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/.

[5] Streamlit Inc. “Streamlit documentation,” Accessed: Nov. 15, 2025. [Online]. Available: https://docs.streamlit.io/.