# Real-time IoT Data Pipeline — Proposal

## Executive summary

We will build a real-time IoT data pipeline that simulates sensor data for temperature, humidity, and CO2. The pipeline will ingest events in real time, process data in batch for cleaning and aggregation, and run streaming checks to generate alerts. A simple dashboard will show live metrics and recent alerts. The generator will write logs in JSONL and CSV and can send events to Azure Event Hubs or to local files.

## Objectives

• Simulate realistic IoT sensor data.

• Provide reliable ingestion using files or Azure Event Hubs.

• Clean and aggregate data in batch.

• Detect and alert on simple anomalies in streaming.

• Provide a simple dashboard for monitoring.

## Scope

In scope:

• Sensor generator in Python that writes JSONL and CSV and can publish to Event Hubs.

• Batch ETL that cleans data, computes aggregates, and flags anomalies.

• Streaming consumer that evaluates alerts.

• Streamlit dashboard to view live metrics and alerts.

Out of scope:

• Full enterprise deployment and advanced production hardening.

• Advanced machine learning models.

## Deliverables

1. sensor\_generator.py with command line options for interval, number of sensors, duration, output folder, and Event Hubs connection.

2. Batch ETL scripts and a notebook that clean data, compute hourly aggregates, and mark anomalies.

3. Streaming consumer that reads events and emits alerts.

4. Streamlit dashboard that shows live metrics and recent alerts.

5. README with run instructions and how to switch between Azure and local modes.

## 3-week timeline

Week 1 — Generator and ingestion: Implement the Python generator, produce sample JSONL and CSV files, and add Event Hubs option.

Week 2 — Batch ETL and storage: Implement data cleaning, compute aggregates, flag anomalies, and store cleaned data in Azure Blob or local files.

Week 3 — Streaming and dashboard: Implement streaming alerts and build a Streamlit dashboard that displays live metrics and recent alerts.

## Tools and how we will use them

• Python 3.10 or higher for the generator, ETL, and streaming code.

• Azure Event Hubs for streaming ingestion.

• Azure Stream Analytics or Azure Functions for streaming rules and alerts.

• Azure Blob Storage or Azure Data Lake to store raw and processed data.

• Pandas and Jupyter for batch ETL and data exploration.

• Streamlit for a lightweight dashboard.

• Git and GitHub for source control.

Fallback options: local files for ingestion, or local Kafka for streaming if Azure is not available.

## Success criteria

• The generator produces valid JSONL and CSV files with timestamps and sensor identifiers.

• The ETL produces cleaned tables and hourly aggregates.

• The streaming job detects threshold breaches and writes alert logs.

• The dashboard shows current metrics and recent alerts.

## Data warehouse design — Star schema

We will use a star schema with one fact table for sensor readings and four dimension tables for time, sensor, location, and status.

ERD diagram (simple):

DIM\_TIME

|

DIM\_LOCATION ---< FACT\_SENSOR\_READING >--- DIM\_SENSOR

|

DIM\_STATUS

### Fact table: fact\_sensor\_reading

Columns:

• reading\_id INT

• time\_id INT

• sensor\_id VARCHAR

• location\_id INT

• status\_id INT

• value FLOAT

• unit VARCHAR

• seq BIGINT

• is\_anomaly BOOLEAN

• anomaly\_type VARCHAR

• ingestion\_ts TIMESTAMP

• processing\_latency\_ms INT

### Dimension table: dim\_time

Columns:

• time\_id INT

• ts TIMESTAMP

• date DATE

• year INT

• month INT

• day INT

• hour INT

• minute INT

• second INT

• day\_of\_week INT

• is\_weekend BOOLEAN

### Dimension table: dim\_sensor

Columns:

• sensor\_id VARCHAR

• sensor\_type VARCHAR

• sensor\_model VARCHAR

• manufacturer VARCHAR

• install\_date DATE

• firmware\_version VARCHAR

• is\_active BOOLEAN

• notes TEXT

### Dimension table: dim\_location

Columns:

• location\_id INT

• site VARCHAR

• zone VARCHAR

• lat FLOAT

• lon FLOAT

• floor VARCHAR

• building VARCHAR

• location\_code VARCHAR

### Dimension table: dim\_status

Columns:

• status\_id INT

• status\_code VARCHAR

• description VARCHAR

## Mapping from JSON/CSV to warehouse

timestamp -> dim\_time.ts and fact\_sensor\_reading.time\_id

sensor\_id -> dim\_sensor.sensor\_id and fact\_sensor\_reading.sensor\_id

sensor\_type -> dim\_sensor.sensor\_type

value -> fact\_sensor\_reading.value

unit -> fact\_sensor\_reading.unit

metadata.location -> dim\_location.location\_code and fact\_sensor\_reading.location\_id

metadata.lat -> dim\_location.lat

metadata.lon -> dim\_location.lon

status -> dim\_status.status\_code and fact\_sensor\_reading.status\_id

seq -> fact\_sensor\_reading.seq

ingestion timestamp -> fact\_sensor\_reading.ingestion\_ts (added by pipeline)

## Example JSON event

{  
 "timestamp": "2025-09-01T17:37:42.379229+03:00",  
 "sensor\_id": "temp\_000",  
 "sensor\_type": "temperature",  
 "value": 20.886,  
 "unit": "C",  
 "metadata": {"lat": 29.9986278, "lon": 31.0047304, "location": "site/zone1", "site\_id": 1, "zone\_id": 1},  
 "status": "OK",  
 "seq": 1,  
 "battery\_level": 94,  
 "firmware\_version": "v1.2.0",  
 "sensor\_model": "T1000",  
 "manufacturer": "AcmeSensors",  
 "signal\_strength": -65,  
 "reading\_quality": 0.98,  
 "is\_simulated": true,  
 "event\_type": "measurement"  
}