GCT CASE STUDY

PROCESS AND DATA ANALYTICS

CO_2 Plant Data Pipeline

Vijay Singh

Date: 14-07-2025

Agenda

Objective

Objective

Design and implement a **real-time data pipeline** that simulates sensor data from a CO_2 capture plant — using **Node-Red** and **AWS native tools** under the **Free Tier**.

The goal was not just to collect data, but to:

- Ingest it in real time
- Structure and process it automatically
- Generate alerts for out-of-range values
- Save the data to another S3 bucket
- Enable insightful dashboards for analysis

Sensor Simulation Design

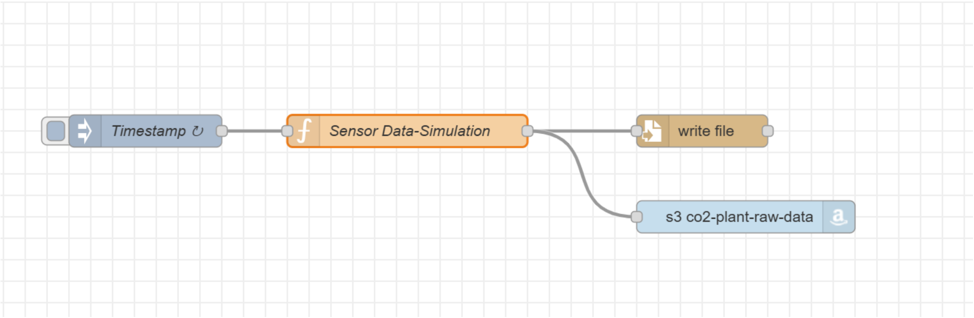
🔧 **Setup Tool: Node-RED (Local Flow)**

Used **Node-RED** to simulate real-time sensor data generation and file delivery.

Sensor Logic

8 Sensors:
pH, conductivity, pressure, temperature, fillLevel, flow, humidity, co2

Frequency:
→ Every **10 seconds**, one reading per sensor
→ Every **1 minute**, 6 readings are saved as one .json file



Node-Red Simulation Design

Data Storage

Dual Output Strategy(for additional data protection)

1. **Local Disk:**
Structured folder hierarchy by Year/Month/Day/Hour

2. **AWS S3:**
JSON files also uploaded directly to co2-plant-raw-data bucket

Folder Structure

🗃️ **Bucket Design**

Raw Data Bucket → co2-plant-raw-data

📁 Folder Structure in S3

Input: `co2-plant-raw-data`

```
raw/
 └── Year YYYY/
      └── Month MM/
           └── Day DD/
                └── HH/
                     └── sensor_HHMM.json
```

Each JSON file contains 6 lines like this:

```
{"timestamp":"...","pH":"7.5",...,"co2":"410.0"}
```

Folder Structure

🗃️ **Bucket Design**

Processed Data Bucket → co2-plant-processed-data

VIJAY SINGH

GCT

Output: `co2-plant-processed-data`

```
processed/
  └── sensor_HHMM_summary.json
```

Each summary file looks like this:

```json
{
  "source_file": "raw/Year.../sensor_0832.json",
  "summary": {
    "pH": {"mean": 7.2, "std_dev": 0.4},
    "temperature": {"mean": 25.3, "std_dev": 1.8}
  },
  "alert_flags": [
    "⚠️ ALERT: pH = 5.8 at 2025-07-12T08:13:40Z"
  ]
}
```

amazon web services S3

pgsql

```
+------------+       (S3 PUT)        +------------+       +-------------+
| Node-RED   | ────────────────────▶ | S3 Bucket  | ──▶   | Lambda      |
| (local)    |                       | (raw data) |       | Processing  |
+------------+                       +------------+       +-------------+
                                                          | - Parse JSON
                                                          | - Aggregate
                                                          | - Alerting
                                                          └────────────▶ Processed S3 Bucket
```

Processing Architecture (AWS Lambda)

Flow Overview

1. Trigger: New file lands in co2-plant-raw-data/raw/

2. Lambda Execution:

- Reads NDJSON file line by line
- Parses readings from 8 sensors
- Calculates:
  - Mean
  - Standard Deviation
- Detects Alert Conditions (pH, temp, CO_2, pressure)
- Saves a processed summary file to co2-plant-processed-data/

GCT

Sample JSONs – Raw vs Processed

Raw Sensor File (NDJSON Format)

Each line = one timestamped reading from all sensors
(6 lines per file, written every minute)


```
{"timestamp":"2025-07-12T23:01:10Z","ph":7.4,"temperature":29.1,"pressure":5.4,"fillLevel":15.9,"
```


Processed Summary File

Created by Lambda after reading all 6 lines

Aggregated + statistically enriched + alert-ready

json


```json
{
  "source_file": "raw/2025/07/12/23/sensor_2328.json",
  "summary": {
    "ph": {"mean": 7.42, "std_dev": 0},
    "temperature": {"mean": 29.12, "std_dev": 0},
    ...
    "co2": {"mean": 412.32, "std_dev": 0}
  },
  "alert_flags": []
}
```

Alert Logic & Thresholds

Thresholds for alerts:

Sensor	Rule
`pH`	`< 6.0 or > 8.5`
`temperature`	`< 5°C or > 40°C`
`pressure`	`< 1 bar or > 10 bar`
`co2`	`< 300 or > 1000 ppm`

json

```json
"alert_flags": [
  "⚠️ ALERT: pH = 5.8 at 2025-07-12T08:13:40Z",
  "⚠️ ALERT: co2 = 1050 at 2025-07-12T08:15:10Z"
]
```

Why we need Alerts?

To detect anomalies and ensure the plant is operating safely and efficiently.
Out-of-range values could signal:

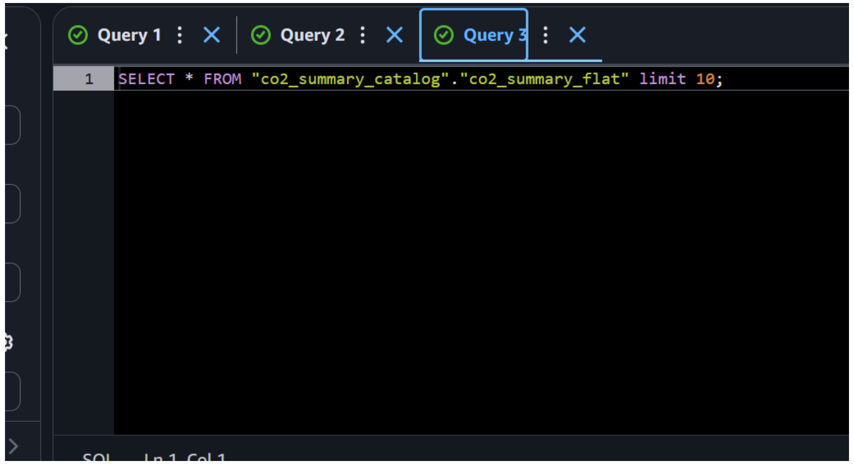
- Sensor malfunction
- Unsafe conditions
- Process inefficiencies

Flexible Design
Rules can be easily extended or modified via config files or database in the future.

Glue

AWS Glue – Data Catalog Setup

- Created a **Crawler** targeting co2-plant-processed-data/processed/
- Automatically infers schema from _summary.json files
- Stores metadata in a Glue **database** (e.g., co2_summary_catalog)
- Runs on schedule or on-demand to reflect new files



Athena + QuickSight

AWS Athena – SQL on JSON

- Connected Athena to the Glue catalog
- Wrote SQL queries to extract sensor stats like:

AWS QuickSight – Dashboard View

- Connected QuickSight to Athena via S3 data source

Visualized metrics like:

- Average temperature and pH trends
- Number of alerts per hour/day
- Comparison charts across sensors

Challenges & Learnings

Category	Challenge
File Timing	Lambda sometimes triggered before S3 file was fully uploaded
Data Format	NDJSON (newline-delimited JSON) caused parsing issues in early versions
IAM Permissions	Needed fine-grained S3 GetObject, PutObject, CloudWatch access
CloudWatch Logs	Lambda invoked, but logs not generated due to missing permissions
Glue Parsing	Nested JSON required schema flattening for Athena/QuickSight

Major Learnings:

1. How to structure cloud storage for **scalability** (date/time folders)

2. Importance of **error handling** in serverless architecture

3. Setting up **fine-tuned IAM roles** for event-driven triggers

4. Power of **Glue + Athena** to query nested, semi-structured JSON

5. Using **Node-RED** for custom IoT pipelines — no external brokers needed

Future Ideas

1. Anomaly Detection (ML-Based)

Use statistical or ML models (e.g., Isolation Forest, Prophet, LSTM) to detect unusual sensor behavior in real time.

2. Time Series Forecasting

Integrate **Amazon Forecast** or **SageMaker** to predict CO_2 trends, pH drift, or equipment behavior.

3. Data Quality Checks

Add automated checks for missing data, duplicates, sensor drift, or outliers.

4. Cost Optimization & Cold Storage

Move older data to **S3 Glacier** to optimize costs.

Thank You!