



DEPARTMENT OF COMPUTER ENGINEERING,  
MODELING, ELECTRONICS AND SYSTEM  
ENGINEERING

LM-Telecommunication Engineering: Smart  
Sensing, Computing and Networking

Distributed Systems and Cloud/Edge Computing

Project Report with Title: Sustainably Smart Facility

Implemented by:

Rami Saleh

Hussein Mohammed

# Introduction:

In today's industrial landscape, ensuring the safety and sustainability of facilities is paramount. Environmental hazards such as temperature surges, humidity imbalances, and gas leaks pose significant risks to both personnel and infrastructure. To address these challenges, the integration of the Internet of Things (IoT) with cloud technologies offers a transformative solution. This project introduces an advanced IoT-enabled system designed to monitor and protect the environmental conditions within industrial facilities. The system leverages advanced sensors and actuators integrated with AWS cloud services to ensure real-time data collection, processing, and response. By utilizing AWS IoT Core, Lambda, DynamoDB, S3, SNS, CloudWatch, Amazon Glue, and Athena, Sustainably Smart Facility demonstrates a robust and scalable approach to protecting the environment within facilities. This data is transmitted securely to the cloud, where powerful analytics and automated workflows ensure immediate responses to potential threats. The system includes intelligent motors, including fans and servo motors, to proactively mitigate risks. This project not only enhances safety and operational efficiency but also aligns with sustainable practices by optimizing resource use and minimizing energy waste. With its ability to detect and respond to environmental risks in real time, this system sets a new standard for protecting industrial environments, ensuring the well-being of both workers and infrastructure.

# Analysis

## Scenario Constraints

**Environment:** The project operates in indoor environments such as facilities, factories, or offices to monitor temperature, humidity, and gas levels to ensure environmental safety.

**Coverage Area:** The system is designed for small to medium-sized facilities, with potential scalability for larger areas by adding more sensors and ESP32 devices.

**Connectivity:** Requires stable Wi-Fi connectivity for real-time communication between ESP32 and AWS IoT Core, Ensures MQTT-based reliable message transmission.

### **Hardware:**

**ESP32 Microcontroller:** Acts as the central device for sensor data collection and actuator control.

**Sensors:** Includes DHT11 for temperature and humidity and MQ2 for gas detection.

**Actuators:** Includes a fan for ventilation and a servo motor for window control.

### **Cloud Infrastructure:**

**AWS IoT Core:** Acts as the primary interface for managing sensor data and sending actuator commands.

**AWS Lambda:** Processes incoming data, applies logic for thresholds, and triggers notifications.

**DynamoDB:** Stores real-time sensor data for structured querying and analysis.

**S3:** Archives sensor data for long-term storage.

**AWS Glue:** Performs ETL (Extract, Transform, Load) operations to prepare data stored in S3 for analysis.

**Athena:** Enables SQL-based querying and analysis of processed data in S3.

**Security:** Secure data transmission with TLS protocols,  
Restricted AWS resource access through IAM policies,  
Encrypted storage for sensitive data.

## **Functional Requirements**

Collect data from DHT11 and MQ2 sensors.

Monitor environmental conditions in real-time.

Trigger actuators (fan and servo motor) based on predefined thresholds.

Send alert notifications when thresholds are breached.

Store sensor data in DynamoDB and S3 for backup and further analysis.

Enable querying and analysis of archived data through AWS Glue and Athena.

## **Non-Functional Requirements**

**Scalability:** Ability to integrate additional sensors and actuators for larger setups.

**Performance:** Real-time processing of sensor data and immediate command responses to actuators.

**Reliability:** Ensure consistent operation with no loss of sensor data or actuator commands.

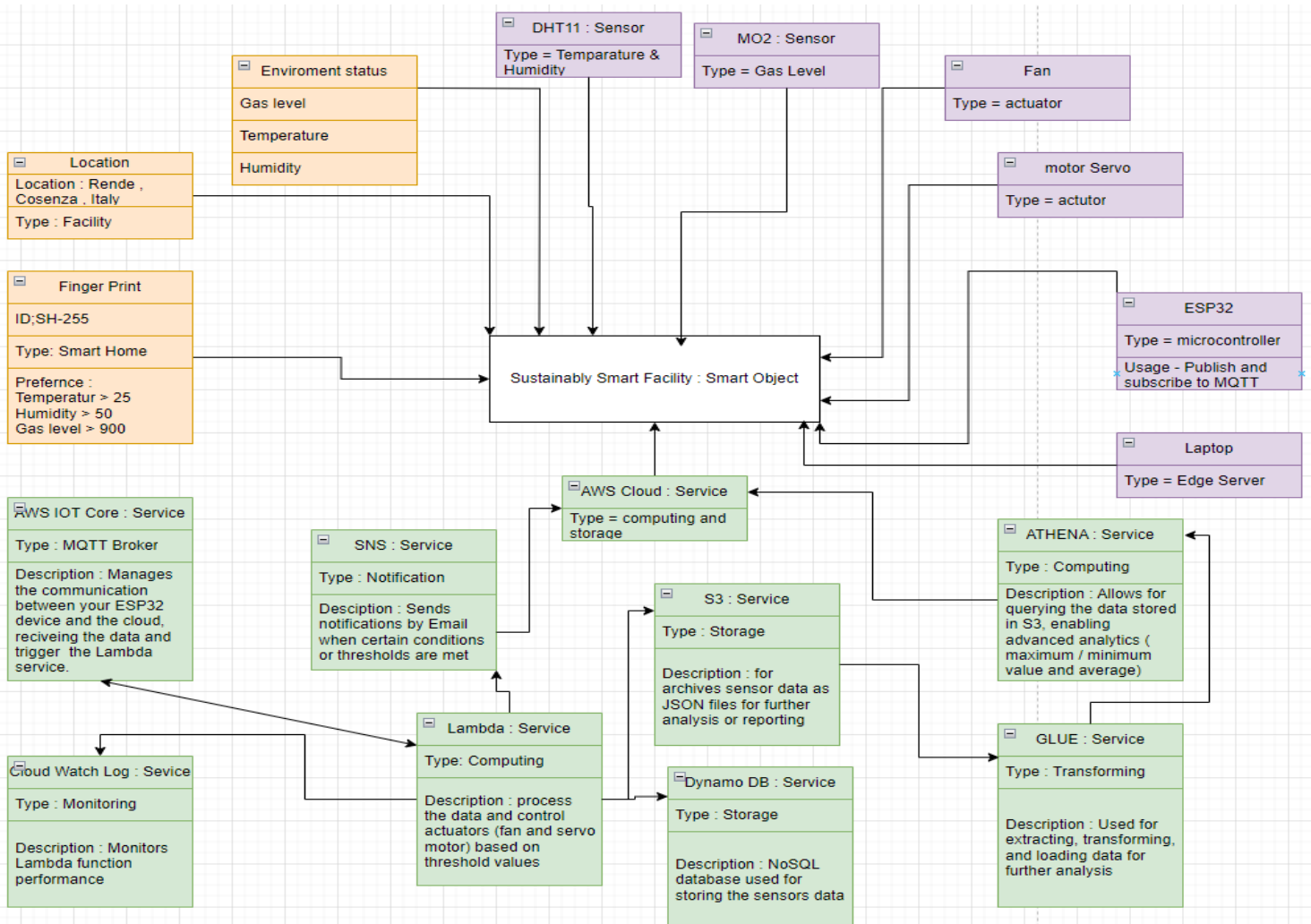
**Security:** Enforce secure communication, access controls, and data integrity.

**Usability:** Provide intuitive dashboards and meaningful notifications.

**Maintainability:** Modular and easy-to-update design for AWS services and firmware.

**Cost Efficiency:** Optimized usage of AWS services to balance cost and functionality.

**Data Integrity:** Ensure accurate storage and retrieval of sensor data for analysis.



**Figure 1: Analysis Model**

# Design

The design choices for "Sustainably Smart Facility" were guided by the project's functional and non-functional requirements, focusing on reliability, scalability, and cost-effectiveness. By leveraging AWS's ecosystem and robust IoT architecture, the design ensures seamless monitoring and control of environmental conditions within the facility.

## Design Choices:

### 1. Data Collection

- **Design Solution:** Use the ESP32
- **Justification:** gathers temperature, humidity, and gas readings and sends them via MQTT to the cloud.

### 2. Device Connectivity

- **Design Solution:** AWS IoT Core
- **Justification:** Receives sensor data and facilitates communication between devices and the cloud.
- **The alternative service is Amazon MQ:** AWS IoT Core is better than Amazon MQ for IoT workloads because it's specifically designed for real-time device communication and offers seamless device integration

### 3. Data Processing

- **Design Solution:** AWS Lambda
- **Justification:** Processes incoming sensor data, checks thresholds, and triggers actuators if needed.
- **The alternative service is AWS Fargate:** Lambda is more cost-effective than Fargate as it charges based on request and execution time, ideal for sporadic workloads and supports efficient serverless execution

#### 4. Data Storage

- **Design Solution:** Amazon DynamoDB + Amazon S3
- **Justification (Dynamo DB):** Stores sensor data for long-term storage.
- **Justification (S3):** Archives sensor data as JSON files for further analysis or reporting.
- **The alternative service (for Dynamo DB) is Amazon Aurora Serverless:** DynamoDB is cheaper for NoSQL workloads compared to Aurora Serverless, which is optimized for relational databases.
- **The alternative service (for S3) is Amazon EFS:** S3 ensures cost-effective scalable storage.

#### 5. Notifications

- **Design Solution:** AWS SNS
- **Justification:** Sends alerts to stakeholders when environmental thresholds are exceeded.
- **The alternative service is Amazon SQS:** SNS Designed for real-time alerts and notifications. SQS requires polling, adding latency and complexity.

#### 6. Monitoring

- **Design Solution:** AWS CloudWatch
- **Justification:** Tracks system performance and logs data for debugging and optimization.
- **The alternative service is AWS X-Ray:** CloudWatch is Better because it is more versatile, integrates seamlessly with all AWS services, and provides both metric monitoring and logging.

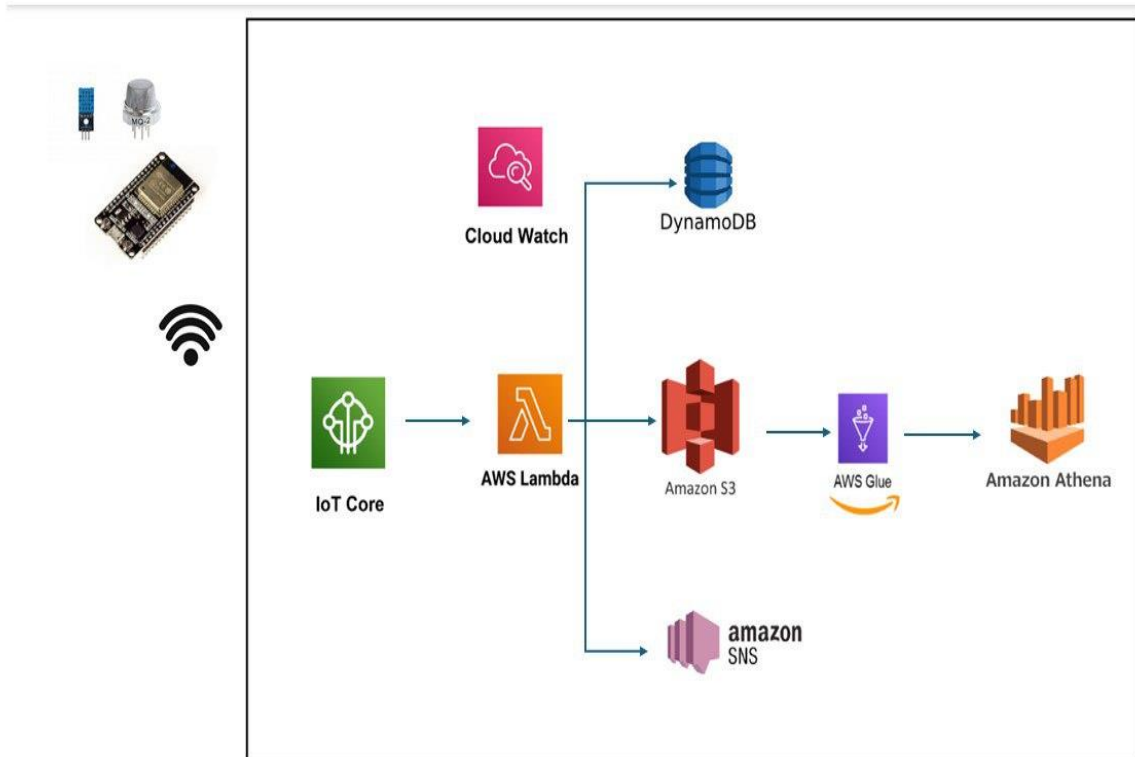
## 7. Data Visualization and Analytics

- **Design solution:** AWS Glue + AWS Athena
- **Justification:** Glue's seamless integration with S3 and Athena's serverless analytics ensure cost-effective, scalable, and efficient data processing and querying without infrastructure management.
- **The alternative service is Amazon Redshift:** AWS Glue & Athena are Cheaper for small-scale analytics and IoT data pipelines. Redshift is better for large datasets but incurs higher costs.

Requirement	Design Solution
Real-Time Data Collection	ESP32 reads sensor data in real-time and processes it using onboard libraries.
Environmental Data Storage	Sensor readings are stored in <b>AWS DynamoDB</b> (real-time data) and <b>AWS S3</b> (archived logs).
Actuator Control Based on Conditions	<b>AWS Lambda</b> triggers actuators (fan and servo motor) based on predefined thresholds from incoming sensor data.
Event Notifications	<b>AWS SNS</b> sends alerts (email/SMS) to stakeholders when gas levels exceed safe thresholds.
Data Visualization and Analytics	Historical data is cleaned using <b>AWS Glue</b> and queried using <b>AWS Athena</b> for analytics.
Reliable Communication	<b>MQTT Protocol</b> ensures lightweight and reliable message exchange between ESP32 and <b>AWS IoT Core</b> .
Security of IoT System	<b>TLS encryption</b> and <b>AWS IoT Policies</b> restrict access to MQTT topics and secure device communications.
Cost-Effective Architecture	<b>The use of AWS serverless services (e.g., Lambda, SNS) minimizes operational costs</b>
System Monitoring and Debugging	<b>AWS CloudWatch Logs:</b> Enables logging for all AWS services, including IoT Core, Lambda, and DynamoDB operations.



# Implementation



**Figure 2: Deployment**

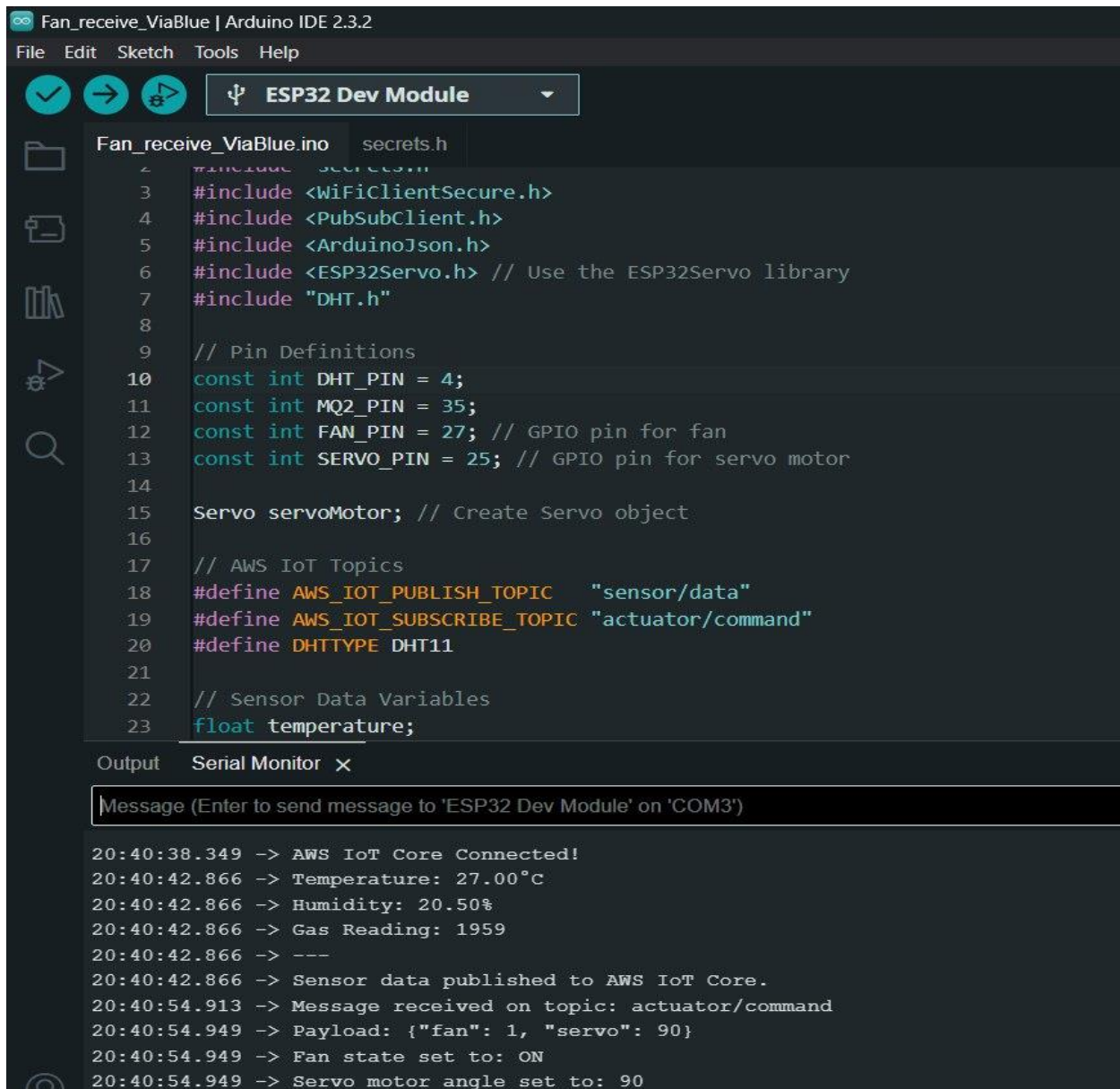
## ESP32

The ESP32 acts as the central hardware, enabling real-time environmental monitoring and automated control within the facility.

it reads data from the MQ2 gas sensor and DHT11 temperature and humidity sensor, then securely transmits this data to the MQTT Broker which is AWS IOT Core and receives actuator commands from AWS IOT Core to control the actuators according to predefined thresholds (The Lambda service will send the commands after comparing the values with

thresholds).

as in the Figure (serial monitor of the Arduino IDE software):



The screenshot displays the Arduino IDE 2.3.2 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for checking, running, and uploading code, along with a dropdown menu for the board type, currently set to 'ESP32 Dev Module'. The main editor area shows the sketch 'Fan\_receive\_ViaBlue.ino' with the following code:

```
1 // Include the necessary libraries
2 #include <WiFiClientSecure.h>
3 #include <PubSubClient.h>
4 #include <ArduinoJson.h>
5 #include <ESP32Servo.h> // Use the ESP32Servo library
6 #include "DHT.h"
7
8 // Pin Definitions
9
10 const int DHT_PIN = 4;
11 const int MQ2_PIN = 35;
12 const int FAN_PIN = 27; // GPIO pin for fan
13 const int SERVO_PIN = 25; // GPIO pin for servo motor
14
15 Servo servoMotor; // Create Servo object
16
17 // AWS IoT Topics
18 #define AWS_IOT_PUBLISH_TOPIC "sensor/data"
19 #define AWS_IOT_SUBSCRIBE_TOPIC "actuator/command"
20 #define DHTTYPE DHT11
21
22 // Sensor Data Variables
23 float temperature;
```

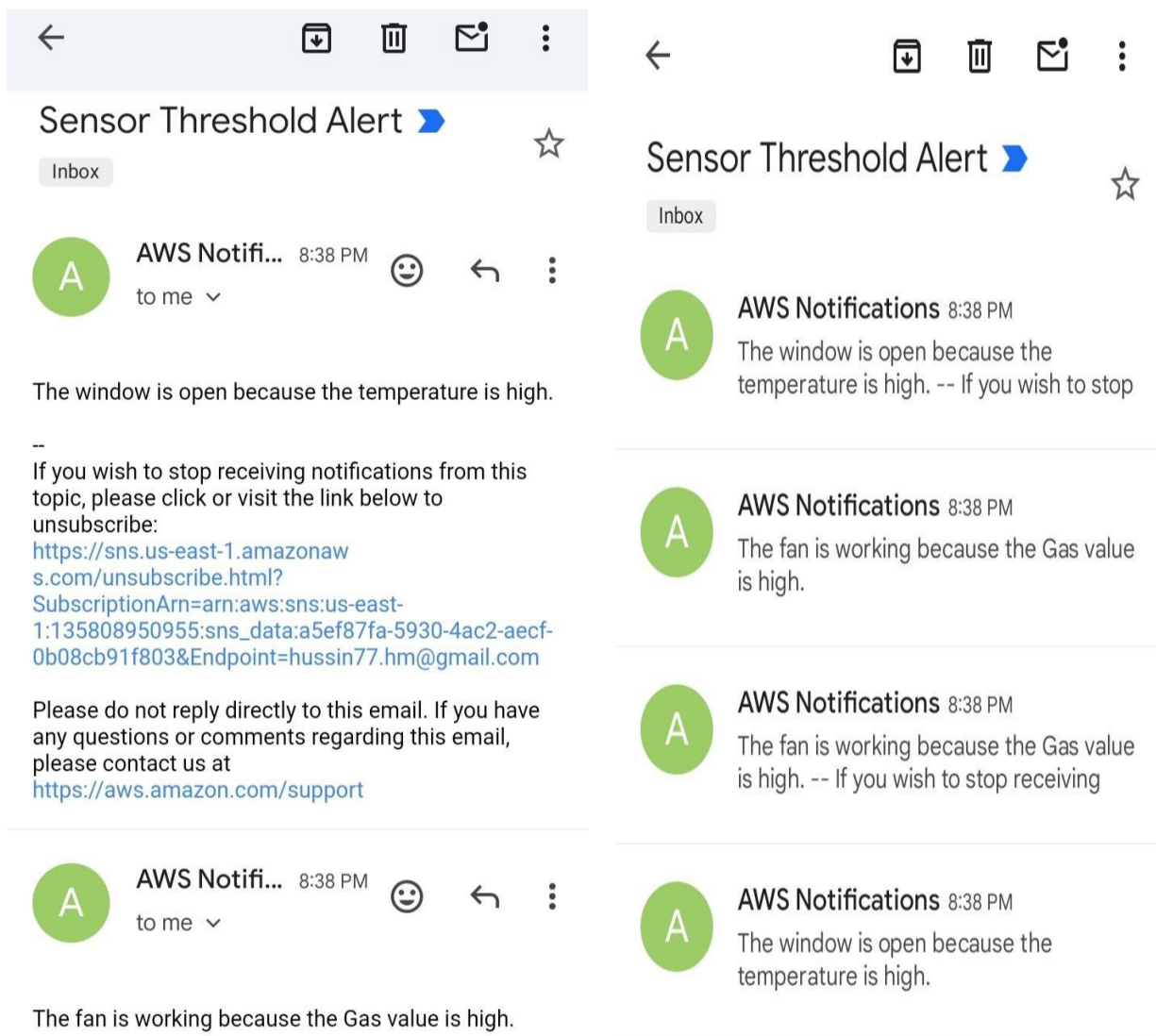
The bottom panel shows the 'Serial Monitor' window, which is currently empty. The status bar at the bottom indicates the board is 'ESP32 Dev Module' and the port is 'COM3'.

**Figure 3: Arduino IDE**

## Lambda Service

The Lambda function serves as the central processing unit of the system, it receives data from AWS IoT Core in the form of temperature, humidity, and gas values and do many actions:

- analysis the incoming data and the compare the values with thresholds and if one or all values exceeded the threshold, the Lambda will:
  - send commands to ESP32 through **AWS IOT Core service** for controlling the actuators.
  - Send notification by Email through **SNS service** (which involves the state of the actuators after one or more values exceeded the threshold).



**Figure 3: Emails**

- Send the values to **Dynamo DB service** which stores structured, real-time sensor data in a NoSQL database
- Send the values to **S3 service** archives sensors data as JSON files for long-term storage and advanced analytics.

## Dynamo DB Service

Amazon DynamoDB serves as a real-time, fully managed NoSQL database to store environmental sensor data.

The table “cloud\_dynamo\_data” consists of:

- a primary key (“sensor\_id” as the partition key)
- a sort key (“timestamp”)

enabling unique identification and chronological querying of sensor readings. Each item includes attributes such as temperature, humidity, gas, sensor\_id, and timestamp.

	sensor_id (String)	timestamp (String)	gas	humidity	temperature
<input type="checkbox"/>	sensor-001	2024-12-30T13:15:54...	1415	22.2	23.2
<input type="checkbox"/>	sensor-001	2024-12-30T13:15:54...	1415	22.2	23.2
<input type="checkbox"/>	sensor-001	2025-01-01T20:58:41...	2033	20	24.3
<input type="checkbox"/>	sensor-001	2025-01-01T20:58:41...	2033	20	24.3
<input type="checkbox"/>	sensor-001	2025-01-01T20:59:10...	2000	20	24.3
<input type="checkbox"/>	sensor-001	2025-01-01T20:59:10...	2000	20	24.3
<input type="checkbox"/>	sensor-001	2025-01-01T21:00:30...	1985	20	24.3
<input type="checkbox"/>	sensor-001	2025-01-01T21:00:30...	1985	20	24.3
<input type="checkbox"/>	sensor-001	2025-01-01T21:01:27...	2000	20	24.2
<input type="checkbox"/>	sensor-001	2025-01-01T21:01:27...	2000	20	24.2

**Figure 4: Dynamo DB**

The table stores scalar types (e.g., strings and numbers) and supports integration with AWS Lambda for automated data ingestion.

## S3 Service

Amazon S3 acts as a reliable and scalable storage solution for sensor data.

The bucket “storequicksight” contains folders such as:

- “sensor\_data/” for JSON files representing sensors readings.
- “athena-results/” for query outputs.

Each JSON file in “sensor\_data/” includes fields like sensor\_id, timestamp, temperature, humidity and gas, storing detailed environmental metrics.

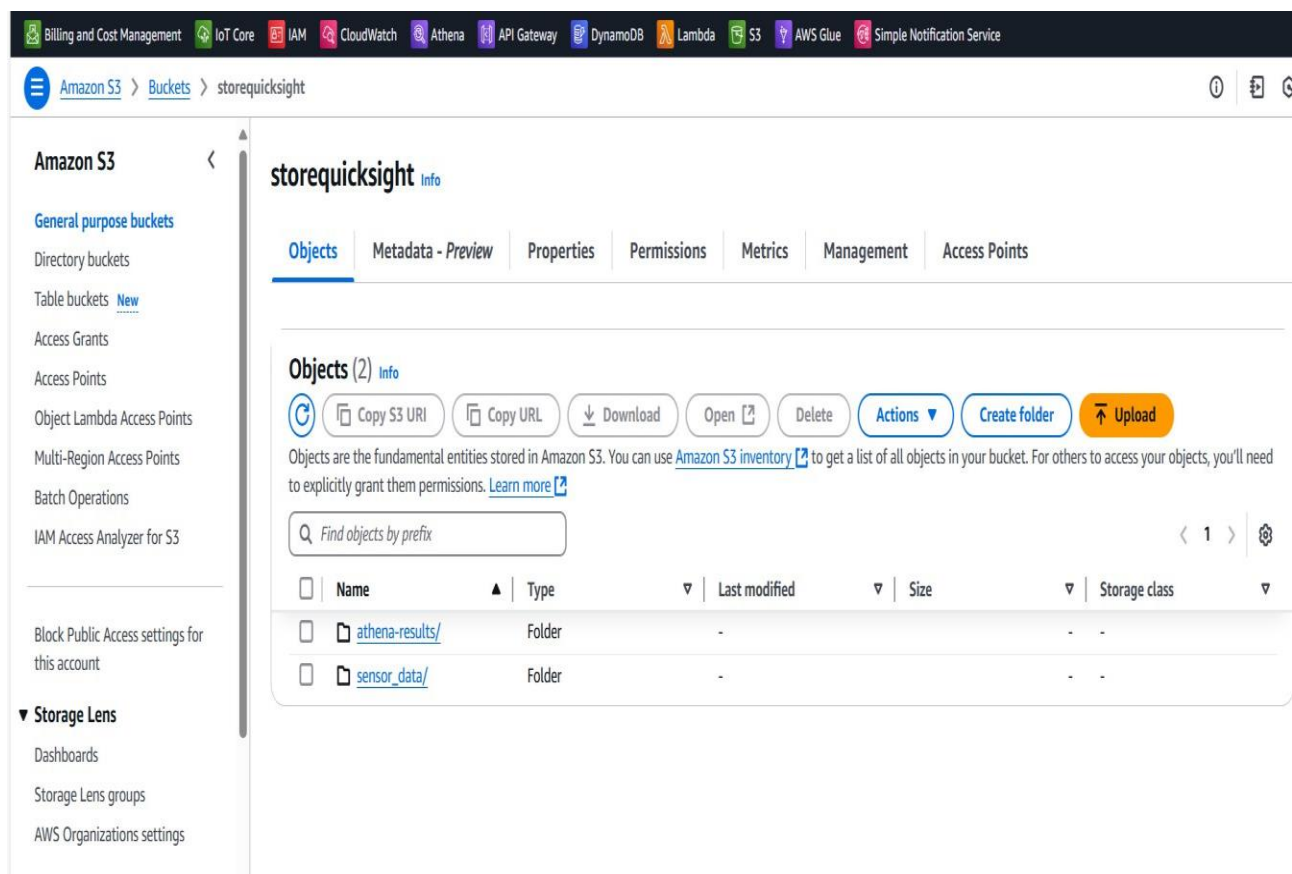


Figure 5: S3

**AWS Glue** integrates with **S3** to prepare the data for analysis. Glue Crawlers scan the “sensor\_data/” folder to detect the schema and create a metadata catalog. This catalog allows **Amazon Athena** to query the sensor data stored in S3 without transferring it to another database.

Athena processes these queries and stores the results in the “athena-results/” folder. This ecosystem enables efficient ETL workflows (Extract, Transform, Load), making raw S3 data accessible for analytics and reporting via Glue and Athena.

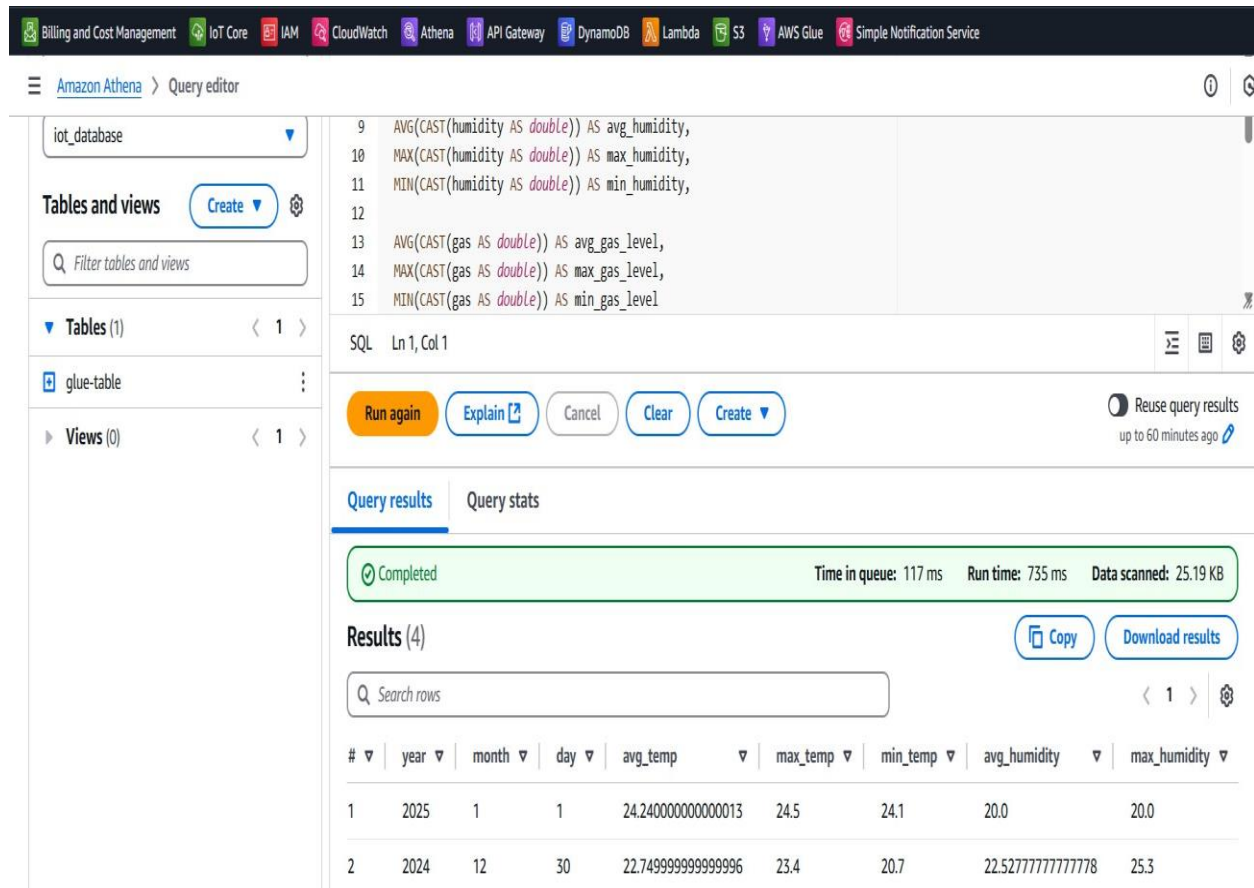
## **Athena Service**

Amazon Athena is a serverless query service for analyzing data directly in **S3** using SQL, eliminating the need for ETL processes or databases.

The Query Editor allows users to write and execute SQL queries efficiently, with seamless integration with the Glue Data Catalog for schema management.

In the project, Athena queries data to generate insights on environmental metrics for example, the query calculates:

- daily averages, maximums, and minimums for temperature, humidity, and gas values by analyzing timestamps and grouping by year, month, and day. This enables efficient data aggregation, trend analysis, and decision-making, with results easily accessible in the S3 folder (“athena-results/”).



**Figure 6: Athena**

## Cloud Watch Log Service

Amazon CloudWatch is a monitoring and observability service that provides real-time metrics, logs, and alerts for AWS services.

In this project, CloudWatch logs record Lambda function activity, including received events, saved data in DynamoDB and S3, sent SNS notifications, and actuator commands.

The logs enable debugging, performance tracking, and error resolution. CloudWatch ensures the seamless operation of all integrated services by offering insights into system behavior.



<a href="#">/aws/lambda/Lambda_Process_Data</a> > 2025/01/01/[\$LATEST]56135782c7a24389a2fd0988c483a2c0		
▶	2025-01-01T21:12:58.427Z	REPORT RequestId: 1562f296-2753-4c54-8fd5-fde6d5977c43 Duration: 576.82 ms
▶	2025-01-01T21:13:27.930Z	START RequestId: 56ea8fbb-bef6-412a-a8ab-9bf861f16d49 Version: \$LATEST
▶	2025-01-01T21:13:27.930Z	Received event: {
▶	2025-01-01T21:13:27.930Z	"temperature": 24.1,
▶	2025-01-01T21:13:27.930Z	"humidity": 20,
▶	2025-01-01T21:13:27.930Z	"gas": 1818
▶	2025-01-01T21:13:27.930Z	}
▶	2025-01-01T21:13:27.950Z	Data saved to DynamoDB.
▶	2025-01-01T21:13:28.459Z	Data saved to S3.
▶	2025-01-01T21:13:28.459Z	Sending alert: The fan is working because the Gas value is high.
▶	2025-01-01T21:13:28.939Z	Notification sent via SNS.
▶	2025-01-01T21:13:28.965Z	Actuator command sent: {'fan': 1, 'servo': 0}
▶	2025-01-01T21:13:28.967Z	END RequestId: 56ea8fbb-bef6-412a-a8ab-9bf861f16d49

**Figure 7: Cloud Watch Log**

## Conclusion

The project successfully demonstrates a robust IoT-based environmental monitoring and control system. By integrating sensors (MQ2 and DHT11) with an ESP32 and leveraging AWS cloud services, it ensures real-time data monitoring, analysis, and responsive control of actuators within the facility. AWS services such as IoT Core, Lambda, DynamoDB, S3, Glue, Athena, and SNS work cohesively to manage sensor data, analyze trends, and trigger notifications or actions based on defined thresholds. This system highlights how IoT and cloud computing can create sustainable and smart facilities to enhance environmental safety, optimize resources, and promote automation.



## **Future Developments**

Future improvements could include integrating machine learning via Amazon SageMaker for predictive analytics, enhancing decision-making capabilities. Adding edge computing with AWS IoT Greengrass could reduce latency and dependency on the cloud. The system could also be expanded to monitor additional environmental parameters, such as air quality (PM2.5/PM10 sensors). Finally, mobile app integration for user-friendly real-time monitoring and system control would significantly enhance usability.