CSS532(Final Project - device Code and Configuration steps)

Name – Sahithi Chimakurthi

Student ID – 2303017

1. Connect the Ultrasonic Sensor to the Raspberry Pi GPIO Pins:

The ultrasonic sensor measures the distance by sending sound waves and calculating the time it takes for the echo to return. For proper operation, connected the sensor as follows:

VCC to the 5V pin on the Raspberry Pi.

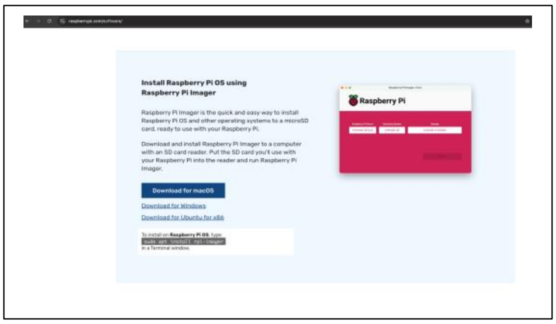
GND to the GND pin on the Raspberry Pi.

TRIG to GPIO Pin 23 on the Raspberry Pi.

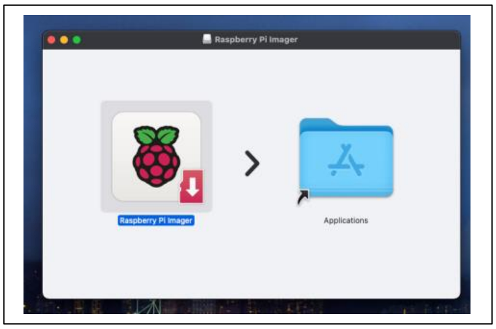
ECHO to GPIO Pin 24 on the Raspberry Pi.

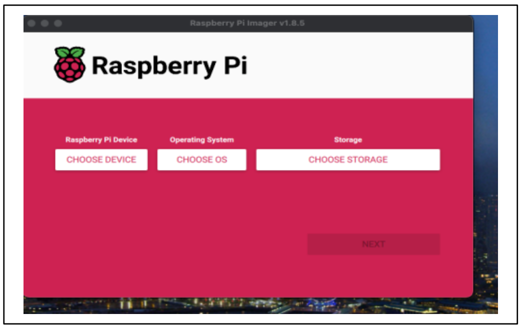
The Raspberry Pi GPIO pins operate at 3.3V, but the sensor operates at 5V. To protect the GPIO pins, I used a 1kΩ resistor and a 2kΩ resistor as a voltage divider for the ECHO pin to step the voltage down to a safe 3.3V before connecting it to the Raspberry Pi

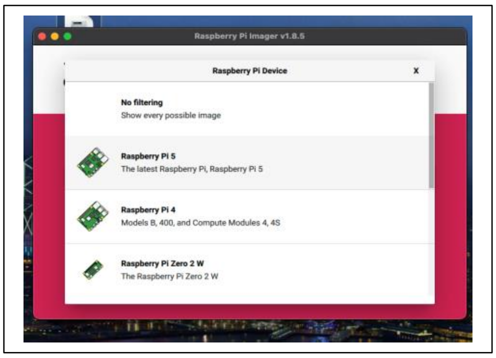
a.) Raspberry Pi Setup: Downloaded and installed the necessary software for the Raspberry Pi.

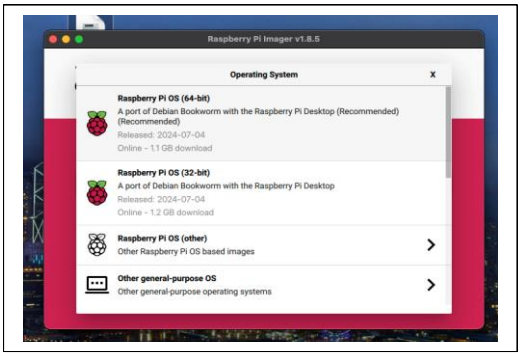


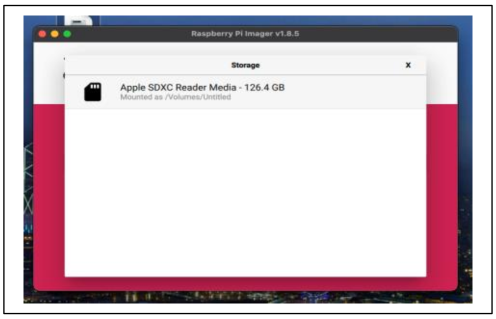
b.) Ran the Raspberry Pi Imager to install the OS onto an external SD card.

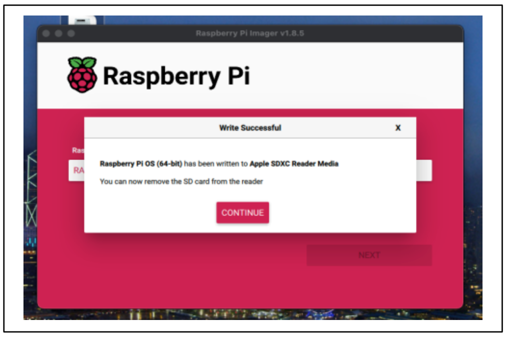




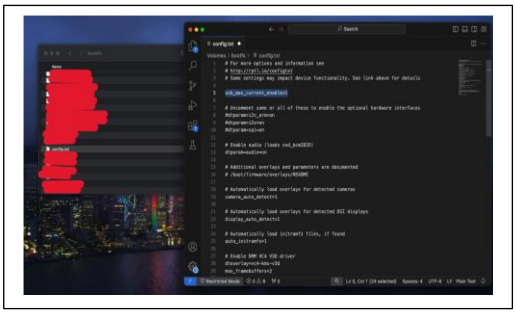




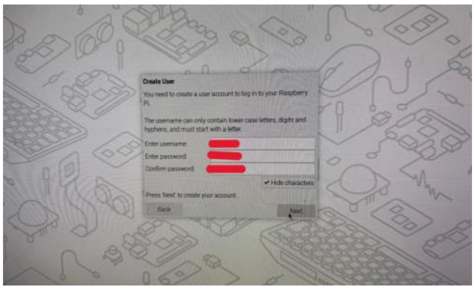


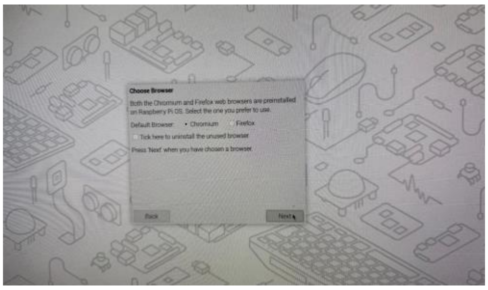


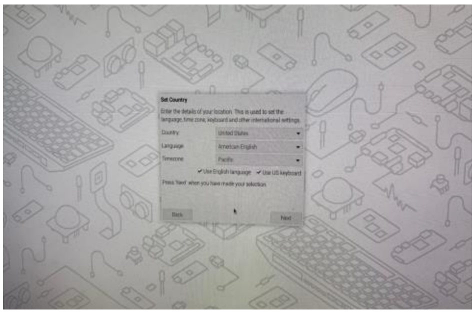
c.) Updated config.txt to enable the Pi to boot from the SD card.

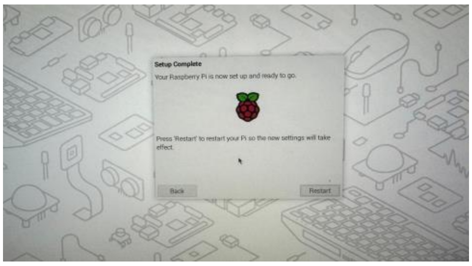


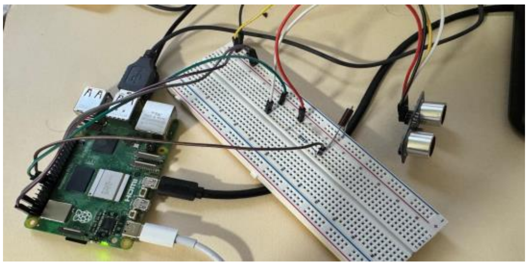
d.) Hardware Setup: Set up the hardware by connecting the Raspberry Pi with the ultrasonic sensor using a breadboard.



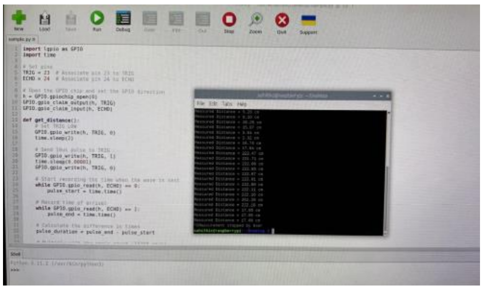
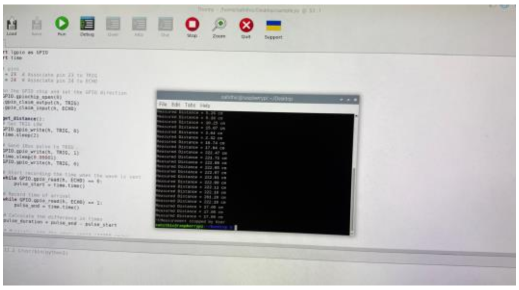




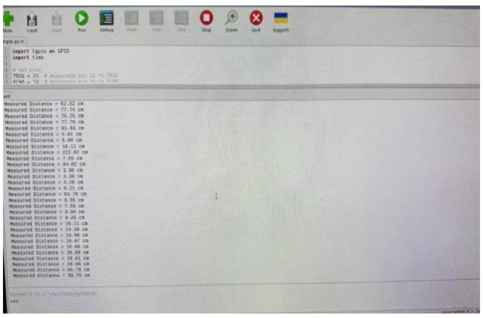




e.) Basic Sensor Functionality: Developed and ran a sample Python script to measure the distance between the sensor and an object for every 2 seconds.

f.) Verified that the distance data outputs correctly in the terminal, confirming the sensor's functionality.



2. Set Up Python Environment and Install Necessary Packages

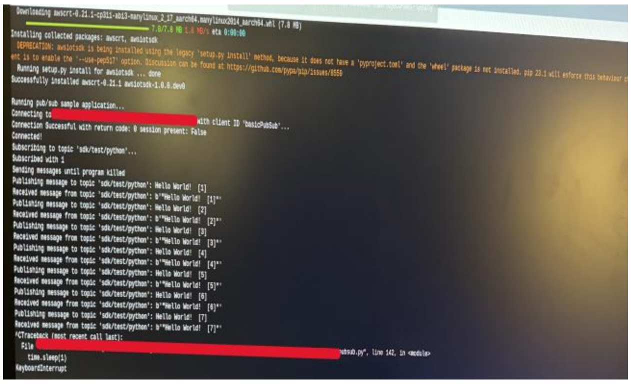
Installed necessary packages like,

paho-mqtt: Library for connecting and publishing messages to MQTT (used in AWS IoT Core).

boto3: AWS SDK for Python to enable Lambda and other AWS interactions.

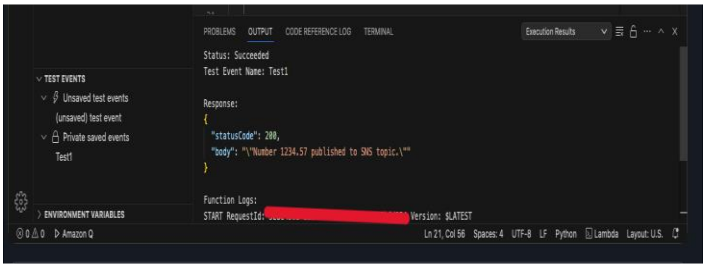
awsiotsdk: AWS IoT SDK to manage device communication.

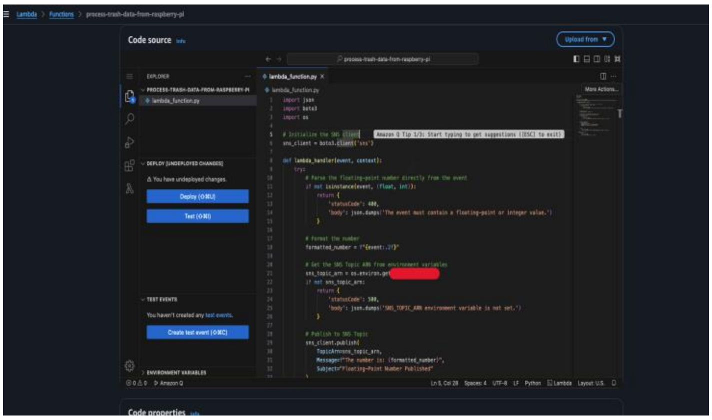
Since newer Raspberry Pi models may restricted direct package installations, I create a new Python environment, activated it and ran a sample script provided by AWS (named start.sh). This script automatically installed all necessary packages and dependencies.



3. Created a Python Script for Measuring Distance

Wrote a sample Python script to send random floating-point numbers to test the MQTT connection and message publishing to AWS IoT Core. This ensured the basic setup works correctly.





After setting Up the AWS IoT Device SDK by configuring Credentials and Connecting to AWS IoT Core, I obtained the necessary credentials from AWS IoT Core like,

a. Certificate to authenticate the device to AWS IoT Core.

b. Private Key which Secures the device's connection.

c. Root CA which Validates the AWS IoT Core server.

I attached an appropriate IoT policy to the device (the "thing") to grant permissions for publishing and subscribing to the MQTT topic (my-Topic).

Then, I configured the device to connect securely to AWS IoT Core using these credentials.

4. Writing the Sensor Distance Script

Then, I created a Python script named sensor_distance.py to measure the distance detected by the ultrasonic sensor. This script Initializes the GPIO pins,

a. To configure the GPIO chip and set the appropriate pin directions for the TRIG and ECHO pins.

b. Sends a pulse to the sensor

c. Sets the TRIG pin to LOW initially

d. Then, sends a precise 10-microsecond pulse to the TRIG pin to trigger the ultrasonic wave

e. Then, records the times by,

Starting the recording time when the wave is sent from the TRIG pin. then, records the time of arrival when the wave is received by the ECHO pin).

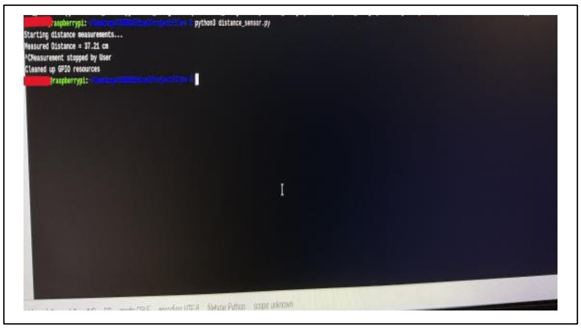
Now, Calculate the distance, by computing the time difference between sending and receiving the wave.

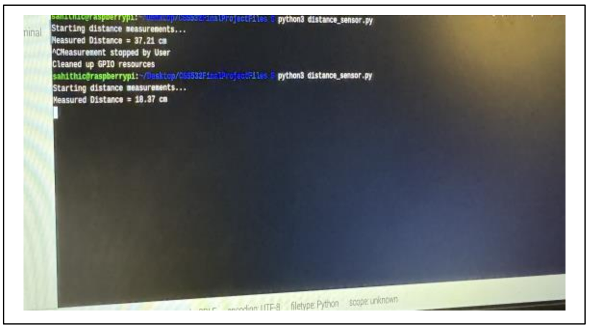
Formula used:

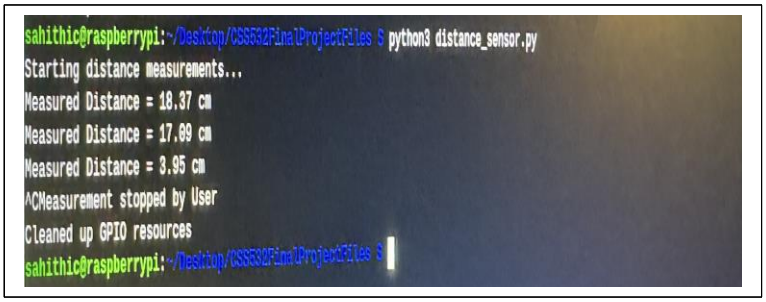
Distance (cm) = (Time Difference × 34300) / 2

(**Note -** The division by 2 accounts for the round trip (to the object and back)).

Then, I stored the calculated distance and the current timestamp in a JSON file, for subsequent processing. Then, repeated this process periodically by measuring the distance at regular intervals (every 5 minutes).



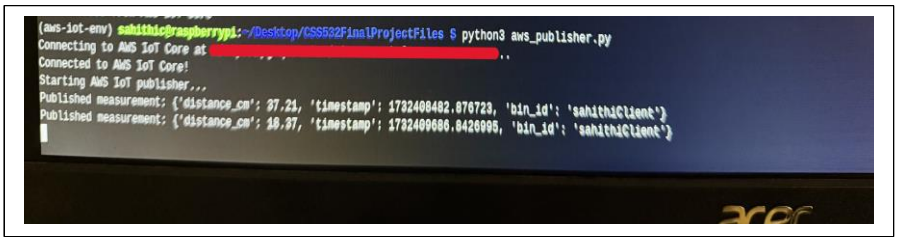




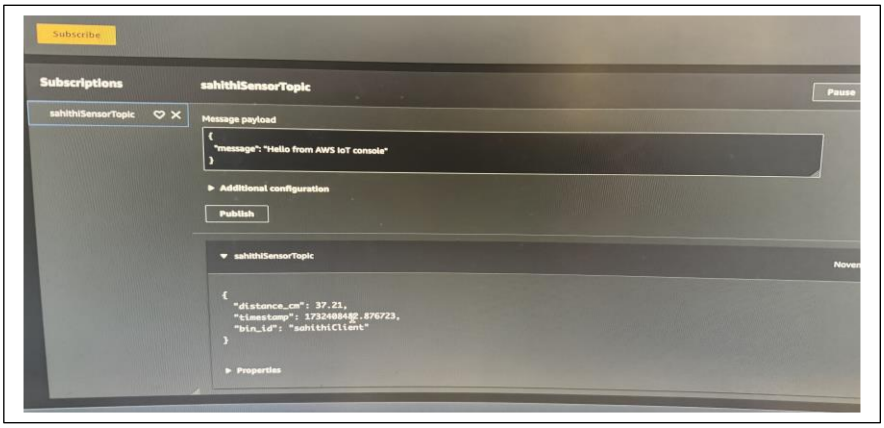
5. Writing the Publishing Script

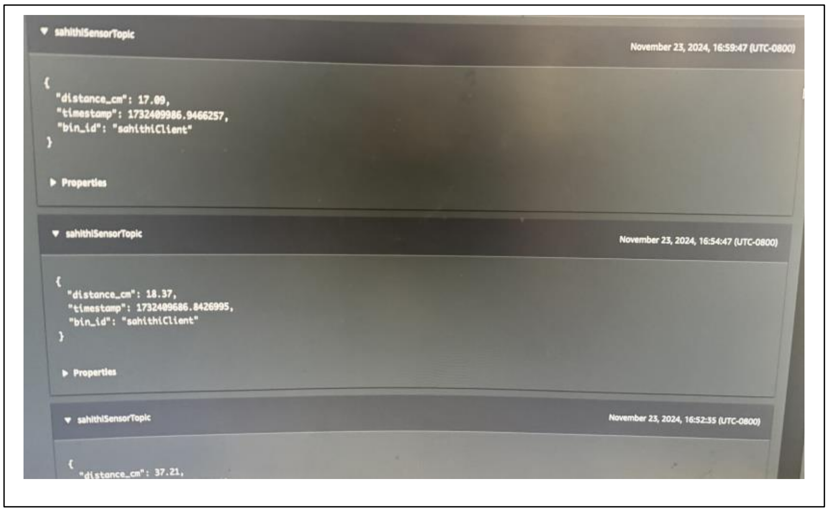
Now, I create a separate Python script named aws_publisher.py to handle data transmission to AWS IoT Core. This script reads the JSON file and

extracts the most recent distance and timestamp data and publishes it to AWS IoT Core. This data is sent to AWS lambda for further processing.



I confirmed that the data was received in the IoT console under the sensor-Topic, and appropriate alerts were triggered.





All notifications were successfully delivered to the email subscribers, ensuring seamless integration and functionality across the system.

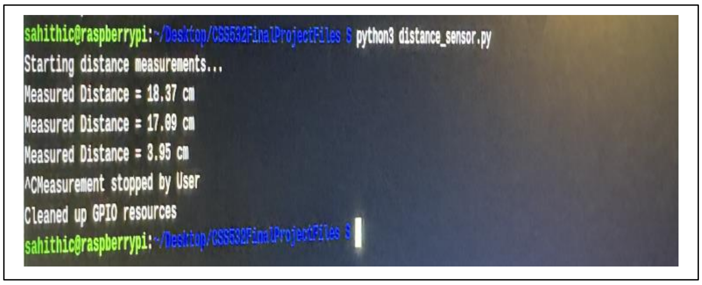
6. Test the sensor’s accuracy:

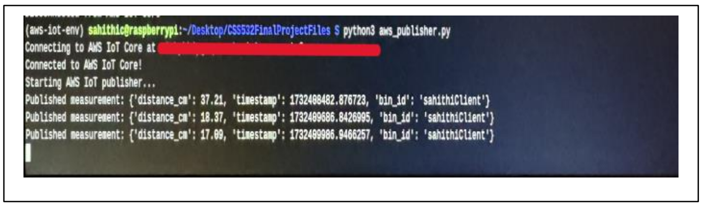
Now, to test the sensor’s accuracy, I ran the sensor script (sensor_distance.py), to verify that the distance is accurately measured and recorded in the JSON file. Also, cross-check the calculated distance with a known measurement to ensure reliability.

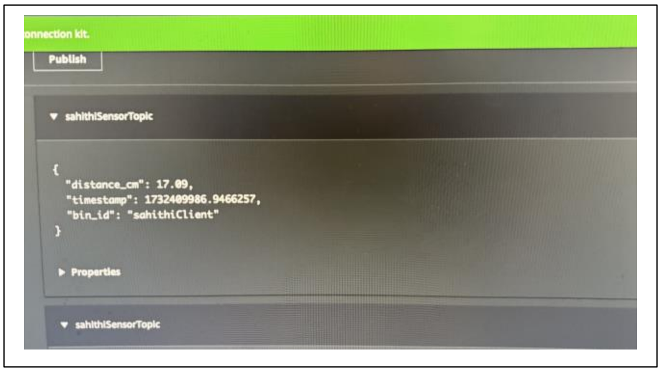
Then, I ran the publishing script (aws_publisher.py) to verify that the data from the JSON file is correctly transmitted to AWS IoT Core.

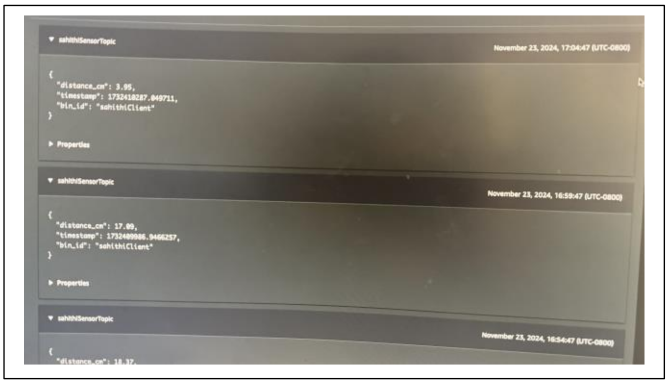
I used the MQTT Test Client in the AWS IoT console to subscribe to myTopic and confirmed the receipt of messages.

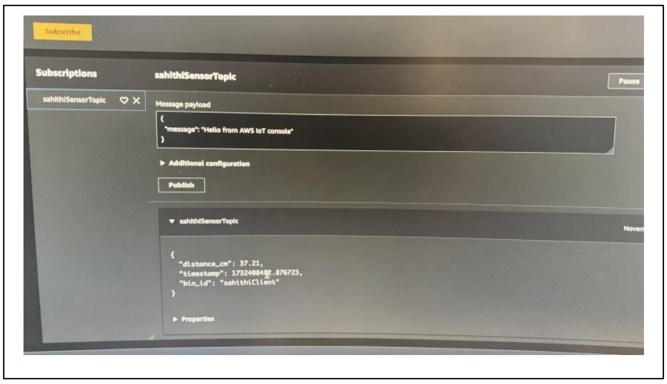
Then, I Continued testing, to observe the consistency and accuracy of distance measurements over multiple cycles. This, ensured that messages appear in AWS IoT Core at 5-minute intervals as expected.











7. Setting Up Data Validation and Logging:

To ensure data accuracy and traceability, I added a validation step in the aws_publisher.py script. Before publishing to AWS IoT Core, the script checks if the calculated distance is within an expected range (e.g., 0 to the height of the bin in centimetres). Any anomalous data points (e.g., negative values or excessively large distances) are logged for review and excluded from publication.

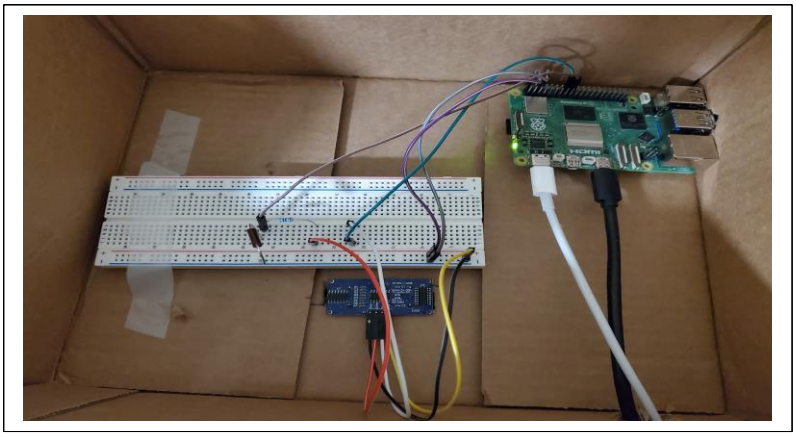
I implemented a logging mechanism using Python's logging library, which captures the details of each data point sent to AWS IoT Core. Logs include a timestamp, the distance measured, and the status of the MQTT message (success/failure). This ensures that issues can be diagnosed efficiently during testing and operation.

(**NOTE:** After completing the setup of AWS IoT Core, Amazon SNS (Simple Notification Service), AWS Lambda Function, AWS IoT Core Rules, Amazon Timestream database, and the Grafana dashboard, I thoroughly tested each component to ensure they were functioning correctly. Upon confirming that all individual components were operating as expected, I proceeded to conduct end-to-end testing of the system to validate the overall functionality of the entire workflow.

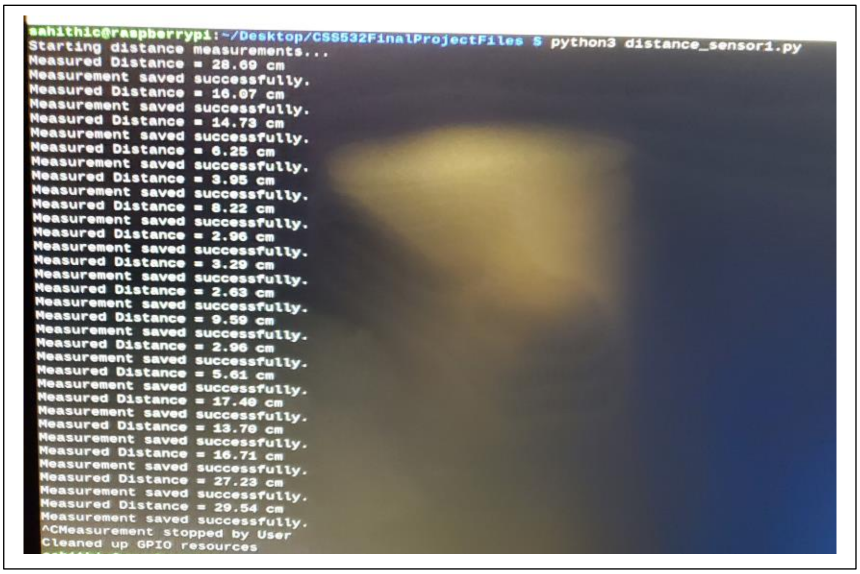
****configuration steps can be found in cloud code configuration file**)**.

7. End-to-End Testing of the Smart Waste Management System:

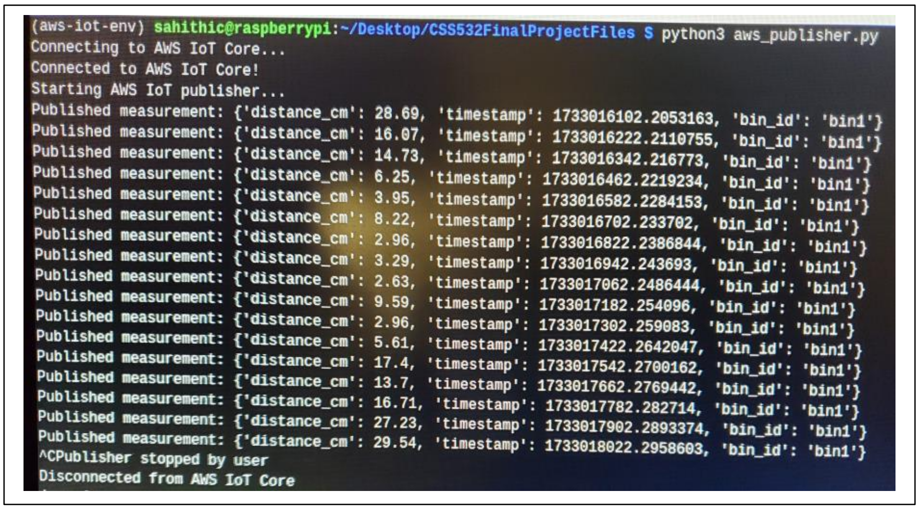
After completing the setup of all components in the smart waste management system, including AWS IoT Core, AWS SNS, AWS Lambda, Amazon Timestream, and the Grafana dashboard, comprehensive end-to-end testing was conducted to validate the system's performance and functionality.



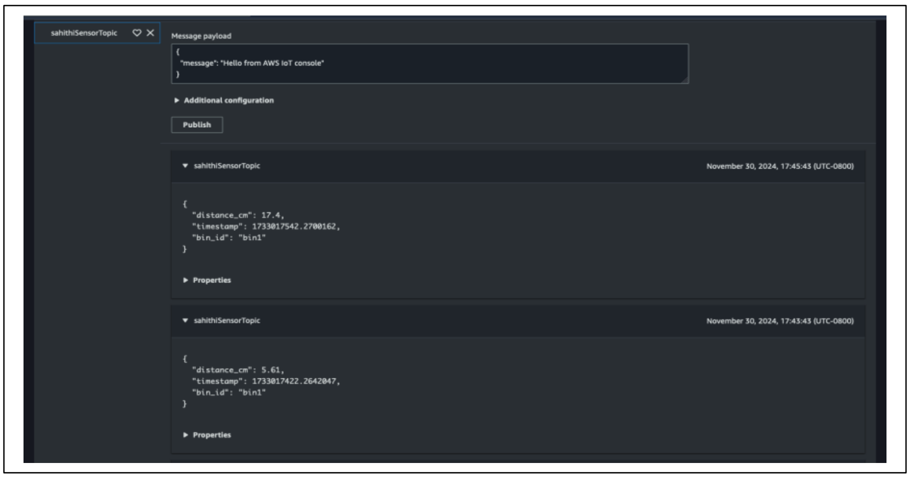
The testing involved running two Python scripts on the Raspberry Pi. The first script interfaced with the ultrasonic sensor, measured the distance to the waste level, and stored the data in a shared JSON file.

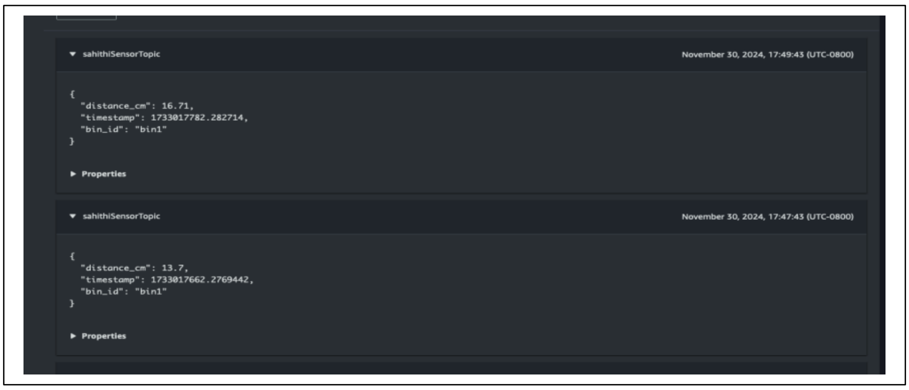


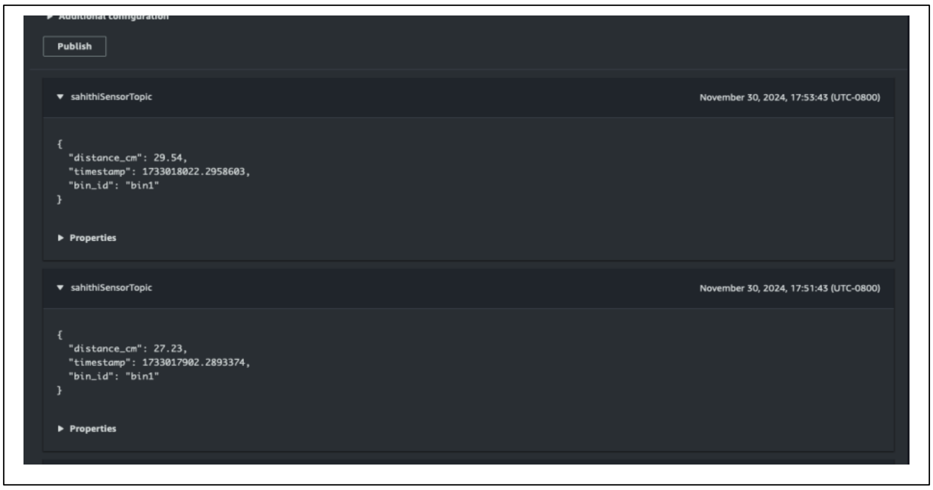
The second script monitored the JSON file continuously for new data entries. When new data was detected, the script published it to AWS IoT Core.


(aws-iot-env) sahithic@raspberrypi:~/Desktop/CSS532FinalProjectFiles $ python3 aws_publisher.py
Connecting to AWS IoT Core...
Connected to AWS IoT Core!
Starting AWS IoT publisher...
Published measurement: {'distance_cm': 28.69, 'timestamp': 1733016102.2053163, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 16.07, 'timestamp': 1733016222.2110755, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 14.73, 'timestamp': 1733016342.216773, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 6.25, 'timestamp': 1733016462.2219234, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 3.95, 'timestamp': 1733016582.2284153, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 8.22, 'timestamp': 1733016702.233702, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 2.96, 'timestamp': 1733016822.2386844, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 3.29, 'timestamp': 1733016942.243693, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 2.63, 'timestamp': 1733017062.2486444, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 9.59, 'timestamp': 1733017182.254096, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 2.96, 'timestamp': 1733017302.259083, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 5.61, 'timestamp': 1733017422.2642047, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 17.4, 'timestamp': 1733017542.2700162, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 13.7, 'timestamp': 1733017662.2769442, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 16.71, 'timestamp': 1733017782.282714, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 27.23, 'timestamp': 1733017902.2893374, 'bin_id': 'bin1'}
Published measurement: {'distance_cm': 29.54, 'timestamp': 1733018022.2958603, 'bin_id': 'bin1'}
^CPublisher stopped by user
Disconnected from AWS IoT Core


Subscriptions
sahithiSensorTopic
Pause
Clear
Export
Edit
sahithiSensorTopic
Message payload
{
"message": "Hello from AWS IoT console"
}
Additional configuration
Publish
sahithiSensorTopic
November 30, 2024, 17:43:43 (UTC-0800)
{
"distance_cm": 5.61,
"timestamp": 1733017422.2642047,
"bin_id": "bin1"
}
Properties
sahithiSensorTopic
November 30, 2024, 17:41:43 (UTC-0800)
{
"distance_cm": 2.96,
"timestamp": 1733017302.259083,
"bin_id": "bin1"
}
Properties


sahithiSensorTopic
Message payload
{
"message": "Hello from AWS IoT console"
}
Additional configuration
Publish
sahithiSensorTopic
November 30, 2024, 17:45:43 (UTC-0800)
{
"distance_cm": 17.4,
"timestamp": 1733017542.2700162,
"bin_id": "bin1"
}
Properties
sahithiSensorTopic
November 30, 2024, 17:43:43 (UTC-0800)
{
"distance_cm": 5.61,
"timestamp": 1733017422.2642047,
"bin_id": "bin1"
}
Properties


sahithiSensorTopic
November 30, 2024, 17:49:43 (UTC-0800)
{
"distance_cm": 16.71,
"timestamp": 1733017782.282714,
"bin_id": "bin1"
}
Properties
sahithiSensorTopic
November 30, 2024, 17:47:43 (UTC-0800)
{
"distance_cm": 13.7,
"timestamp": 1733017662.2769442,
"bin_id": "bin1"
}
Properties



(**Note:** I was unable to capture all the messages published to AWS IoT Core as the system logged me out every 15 minutes.)

AWS IoT Core redirected the data to AWS Lambda, which processed the information to compute the bin's fill level, stored it in Amazon Timestream along with a timestamp, and triggered alerts based on defined thresholds.

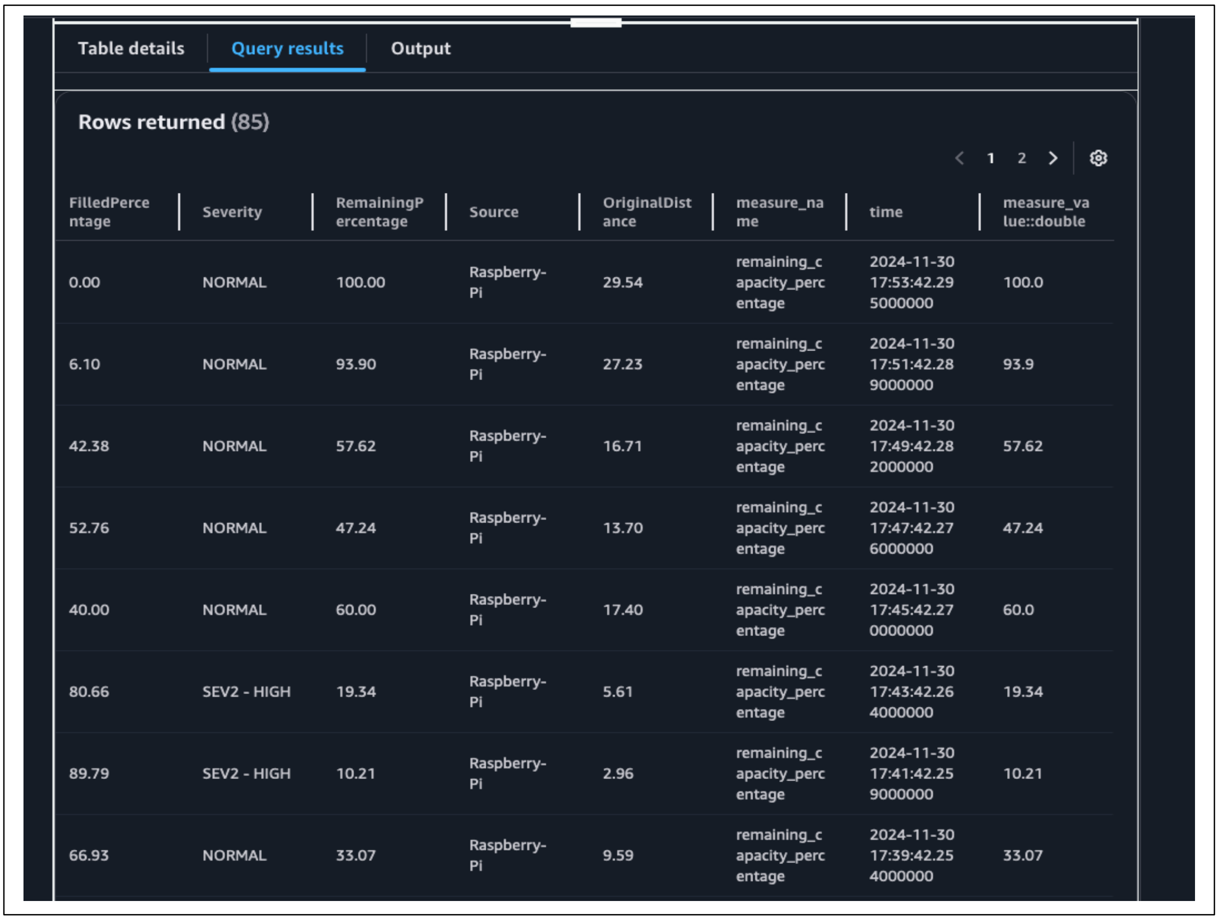
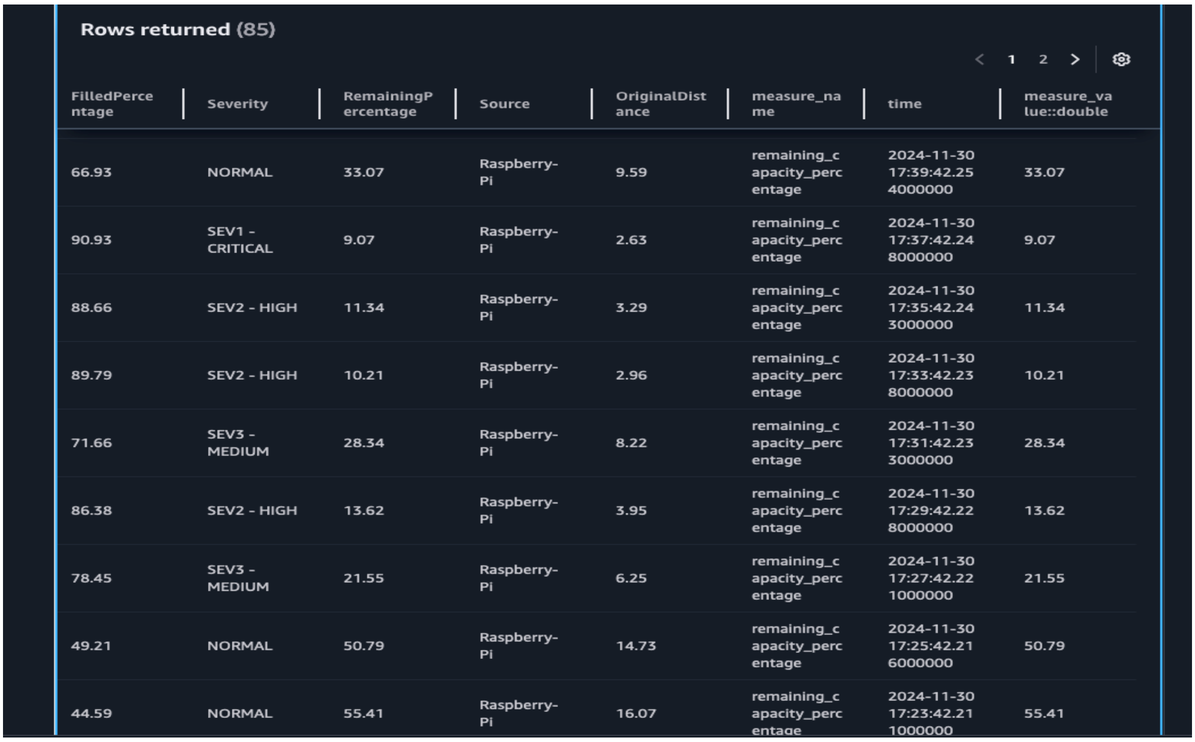


Table details | Query results | Output

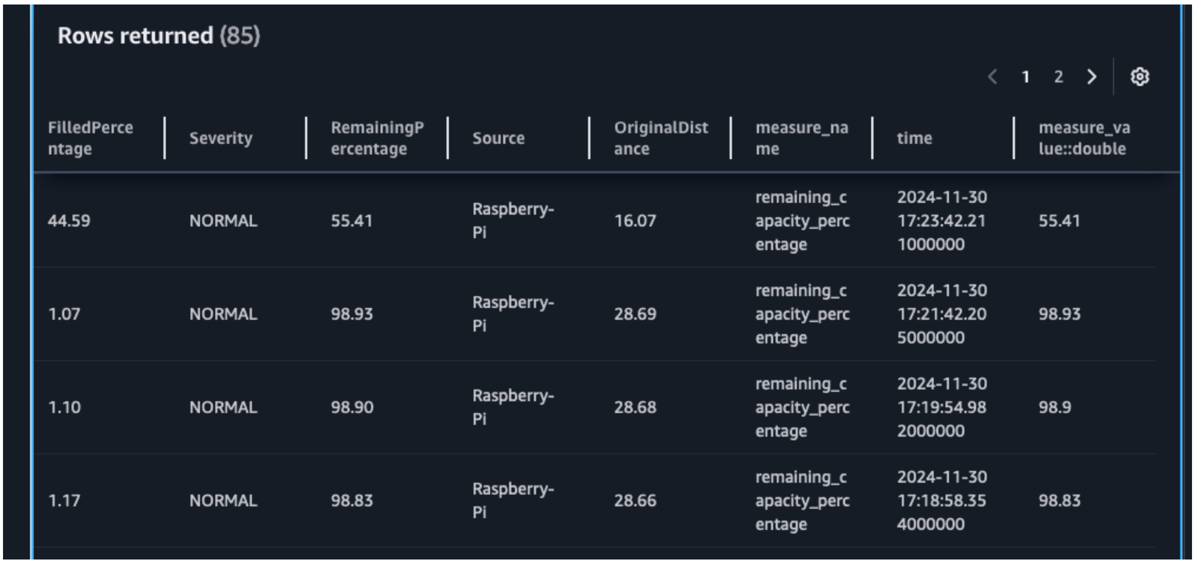
Rows returned (85)

FilledPercentage	Severity	RemainingPercentage	Source	OriginalDistance	measure_name	time	measure_value::double
0.00	NORMAL	100.00	Raspberry-Pi	29.54	remaining_capacity_percentage	2024-11-30 17:53:42.295000000	100.0
6.10	NORMAL	93.90	Raspberry-Pi	27.23	remaining_capacity_percentage	2024-11-30 17:51:42.289000000	93.9
42.38	NORMAL	57.62	Raspberry-Pi	16.71	remaining_capacity_percentage	2024-11-30 17:49:42.282000000	57.62
52.76	NORMAL	47.24	Raspberry-Pi	13.70	remaining_capacity_percentage	2024-11-30 17:47:42.276000000	47.24
40.00	NORMAL	60.00	Raspberry-Pi	17.40	remaining_capacity_percentage	2024-11-30 17:45:42.270000000	60.0
80.66	SEV2 - HIGH	19.34	Raspberry-Pi	5.61	remaining_capacity_percentage	2024-11-30 17:43:42.264000000	19.34
89.79	SEV2 - HIGH	10.21	Raspberry-Pi	2.96	remaining_capacity_percentage	2024-11-30 17:41:42.259000000	10.21
66.93	NORMAL	33.07	Raspberry-Pi	9.59	remaining_capacity_percentage	2024-11-30 17:39:42.254000000	33.07



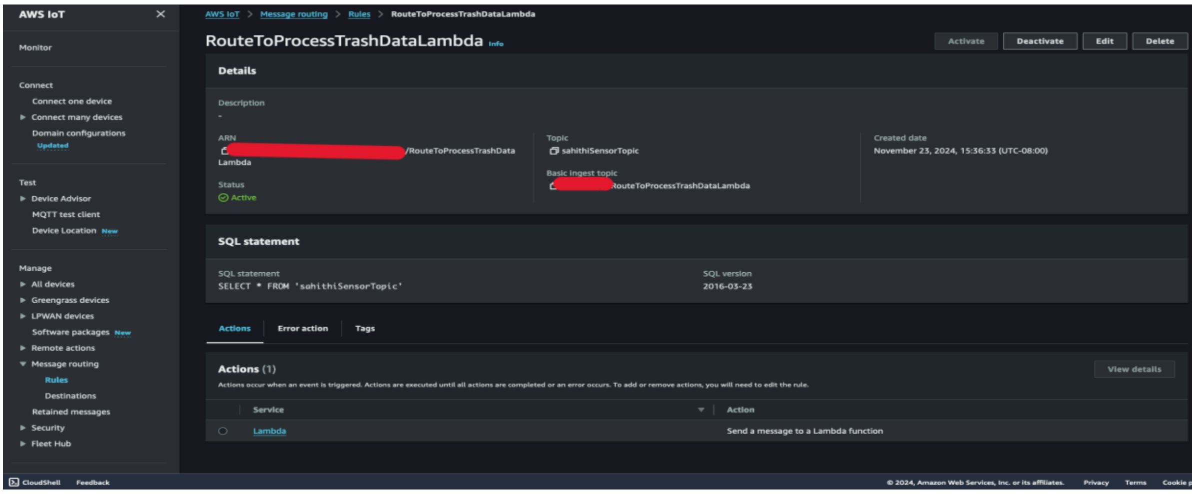
Rows returned (85)

FilledPercentage	Severity	RemainingPercentage	Source	OriginalDistance	measure_name	time	measure_value::double
66.93	NORMAL	33.07	Raspberry-Pi	9.59	remaining_capacity_percentage	2024-11-30 17:39:42.254000000	33.07
90.93	SEV1 - CRITICAL	9.07	Raspberry-Pi	2.63	remaining_capacity_percentage	2024-11-30 17:37:42.248000000	9.07
88.66	SEV2 - HIGH	11.34	Raspberry-Pi	3.29	remaining_capacity_percentage	2024-11-30 17:35:42.243000000	11.34
89.79	SEV2 - HIGH	10.21	Raspberry-Pi	2.96	remaining_capacity_percentage	2024-11-30 17:33:42.238000000	10.21
71.66	SEV3 - MEDIUM	28.34	Raspberry-Pi	8.22	remaining_capacity_percentage	2024-11-30 17:31:42.233000000	28.34
86.38	SEV2 - HIGH	13.62	Raspberry-Pi	3.95	remaining_capacity_percentage	2024-11-30 17:29:42.228000000	13.62
78.45	SEV3 - MEDIUM	21.55	Raspberry-Pi	6.25	remaining_capacity_percentage	2024-11-30 17:27:42.221000000	21.55
49.21	NORMAL	50.79	Raspberry-Pi	14.73	remaining_capacity_percentage	2024-11-30 17:25:42.216000000	50.79
44.59	NORMAL	55.41	Raspberry-Pi	16.07	remaining_capacity_percentage	2024-11-30 17:23:42.211000000	55.41

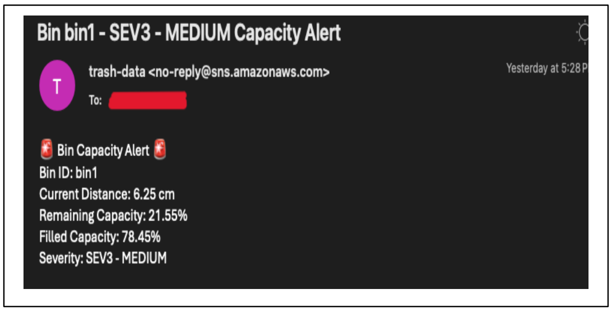


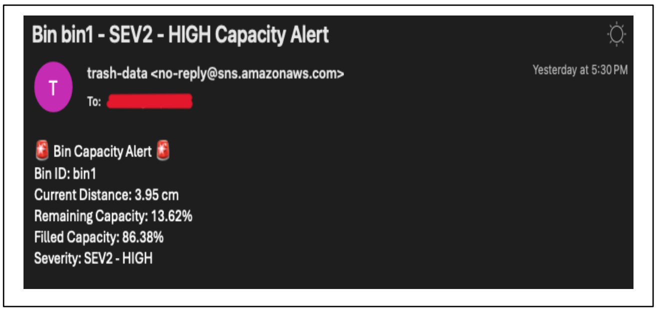
Rows returned (85)

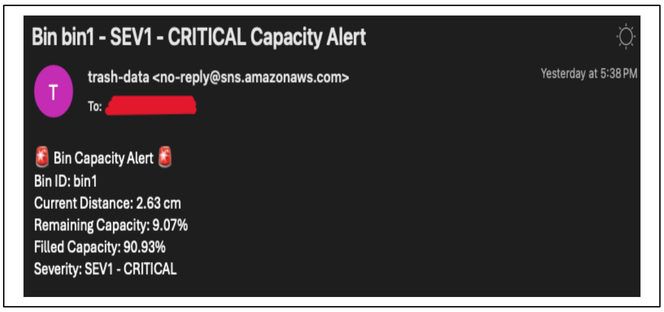
FilledPercentage	Severity	RemainingPercentage	Source	OriginalDistance	measure_name	time	measure_value::double
44.59	NORMAL	55.41	Raspberry-Pi	16.07	remaining_capacity_percentage	2024-11-30 17:23:42.211000000	55.41
1.07	NORMAL	98.93	Raspberry-Pi	28.69	remaining_capacity_percentage	2024-11-30 17:21:42.205000000	98.93
1.10	NORMAL	98.90	Raspberry-Pi	28.68	remaining_capacity_percentage	2024-11-30 17:19:54.982000000	98.9
1.17	NORMAL	98.83	Raspberry-Pi	28.66	remaining_capacity_percentage	2024-11-30 17:18:58.354000000	98.83

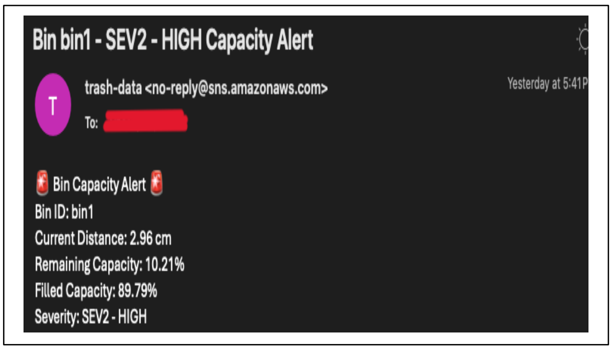


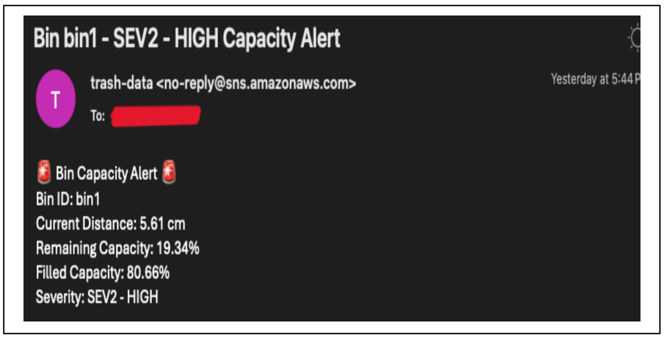
The Grafana dashboard was configured to visualize the data, with the x-axis representing time and the y-axis showing the percentage of remaining bin space. Testing began with an empty bin, and trash was added and removed randomly to simulate dynamic waste accumulation. The system accurately measured and updated the bin's fill level throughout the test. Threshold-based alerts were generated correctly: at 70% (Sev3), 80% (Sev2), and 90% (Sev1) fill levels, demonstrating the efficacy of the alerting mechanism.



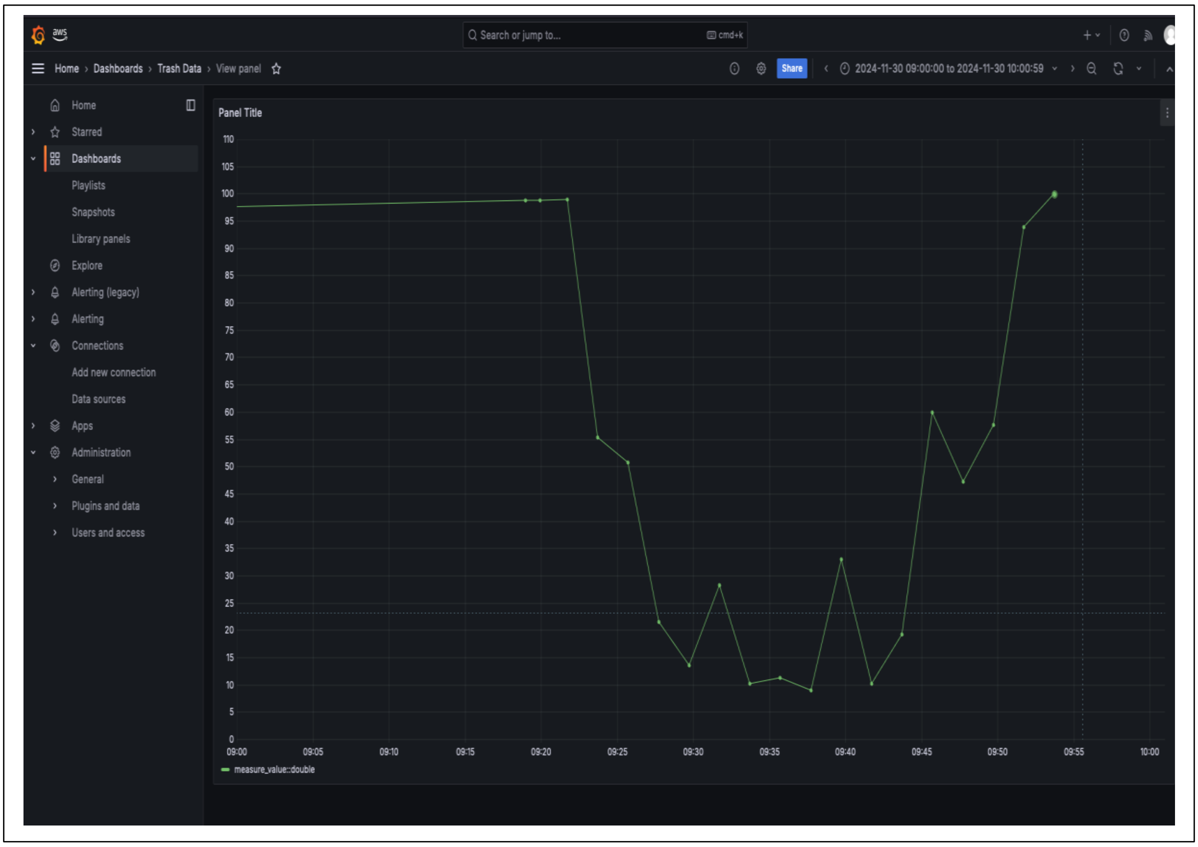


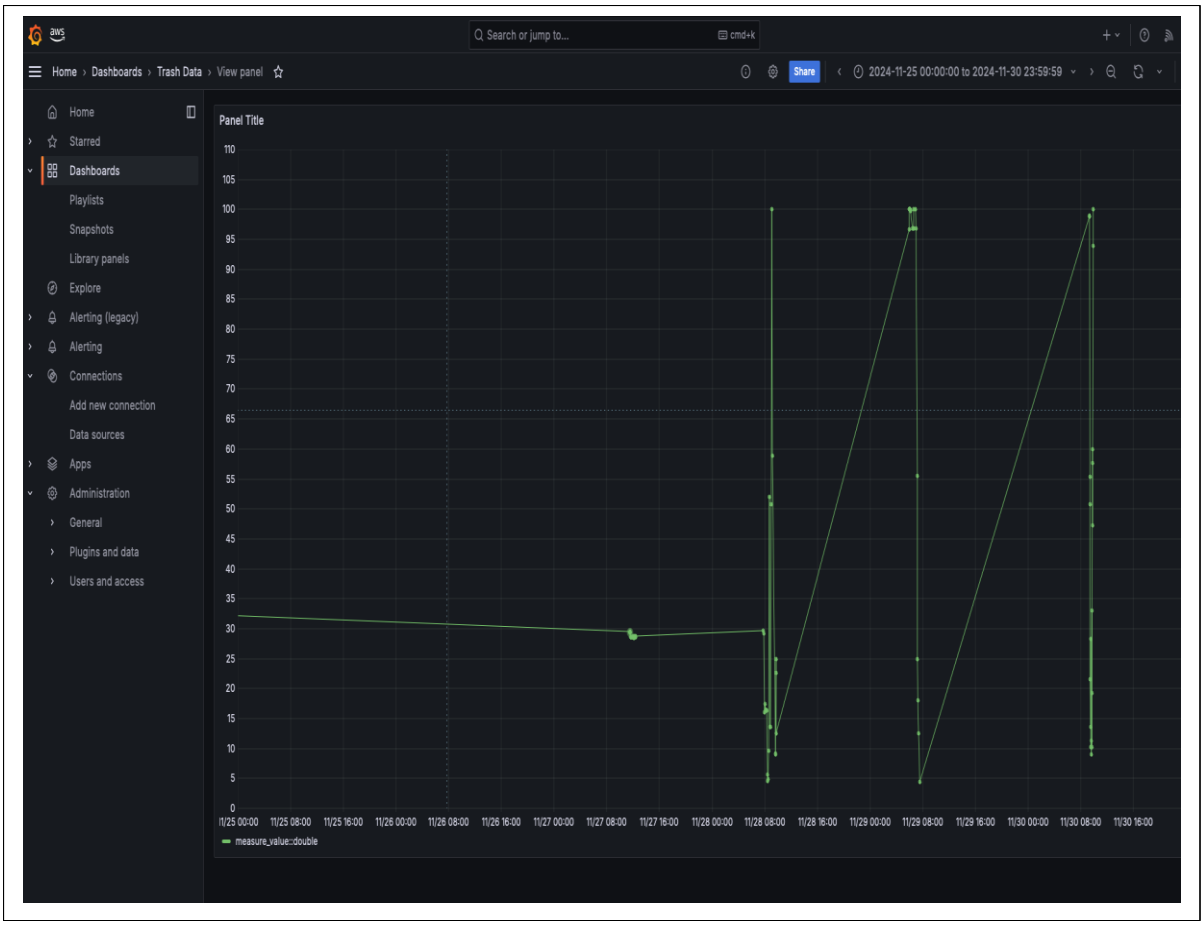






The data flow across the system was monitored at each stage. Measurements were successfully transmitted to AWS IoT Core, processed by AWS Lambda, stored in Amazon Timestream, and displayed visually in Grafana in real-time. The system consistently demonstrated accurate distance measurement, timely alert generation, and efficient data visualization, validating its reliability and suitability for practical waste management applications.




Search or jump to...
cmd+k
Home › Dashboards › Trash Data › View panel
Share
2024-11-25 00:00:00 to 2024-11-30 23:59:59
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Explore
Alerting (legacy)
Alerting
Connections
Add new connection
Data sources
Apps
Administration
General
Plugins and data
Users and access
Panel Title
110
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
11/25 00:00
11/25 08:00
11/25 16:00
11/26 00:00
11/26 08:00
11/26 16:00
11/27 00:00
11/27 08:00
11/27 16:00
11/28 00:00
11/28 08:00
11/28 16:00
11/29 00:00
11/29 08:00
11/29 16:00
11/30 00:00
11/30 08:00
11/30 16:00
measure_value::double


Search or jump to...
cmd+k
Home › Dashboards › Trash Data › View panel
Share
2024-11-28 00:00:00 to 2024-11-30 23:59:59
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Explore
Alerting (legacy)
Alerting
Connections
Add new connection
Data sources
Apps
Administration
General
Plugins and data
Users and access
Panel Title
110
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
11/28 00:00
11/28 04:00
11/28 08:00
11/28 12:00
11/28 16:00
11/28 20:00
11/29 00:00
11/29 04:00
11/29 08:00
11/29 12:00
11/29 16:00
11/29 20:00
11/30 00:00
11/30 04:00
11/30 08:00
11/30 12:00
11/30 16:00
11/30 20:00
measure_value::double

Encountered challenges:

The development of the device code and integration with cloud services presented multiple challenges that required iterative troubleshooting and adaptations to resolve, some of them are

1. Sensor Calibration: The ultrasonic sensor initially produced inconsistent readings due to variations in sensitivity across different surfaces. This issue was resolved by conducting tests under diverse conditions and recalibrating the sensor to ensure accurate measurements in real-world scenarios.

2. Raspberry Pi Booting Issues: While setting up the Raspberry Pi, the device failed to boot from the external SD card. This was addressed by carefully re-flashing the SD card and making necessary adjustments to the config.txt file to enable proper booting.

3. AWS IoT Core Connection Timeout: While testing the data publishing functionality, I faced periodic disconnections from AWS IoT Core due to session timeouts, which occurred approximately every 15 minutes. This required frequent re-authentication, which posed a challenge for continuous monitoring.

Time spent to work on device code:

I spent nearly ~ 40 hours for device code in the past 1 month.